P9 IPL Flow

v1.08 (05/01/17)

Copyright and Disclaimer

IBM Systems and Technology Group 2070 Route 52, Bldg. 330 Hopewell Junction, NY 12533-6351

1 Introduction

1.1 Description

This document will describe the high-level IPL flow for the Power servers based on P9 chips. It is not intended to contain all low-level details, but instead is designed to illustrate the relationships between various low-level procedures. Complete details can be found in the reference documents listed at the end of this document.

This document covers both the hardware and firmware flow required to boot a system to the hypervisor state. This includes full energy management capability and enough resources to boot partitions. Historical procedure names will continue to be used to identify distinct function boundaries, but actual format may vary.

Note on accuracy: All details down to the procedure call are correct and this document is considered an authoritative reference. Any details within an individual procedure are informational only, the final authority lies within the procedures themselves and their associated reference documents.

This version of the document will cover all POWER8 systems. Please note that this document has a lot of low level details on the initialization of the POWER processor and it's memory subsystem. There are a lot of terms and details in here that are very IBM and POWER centric. We attempted to put as much in the glossary as possible but please feel free to use the mailing list for any questions.

Throughout the document you will see references to a “SP”. This stands for a service processor and when used it's applicable to either the FSP (Flexible Service Processor – used within IBM POWER based servers) or the BMC (the OpenPower service processor). For the most part we've tried to remove FSP specific references for the OpenPower work but some may still remain for reference in here.

Reading over the Hostboot Programmers guide (same document repo) is reccomended prior to reading this document.

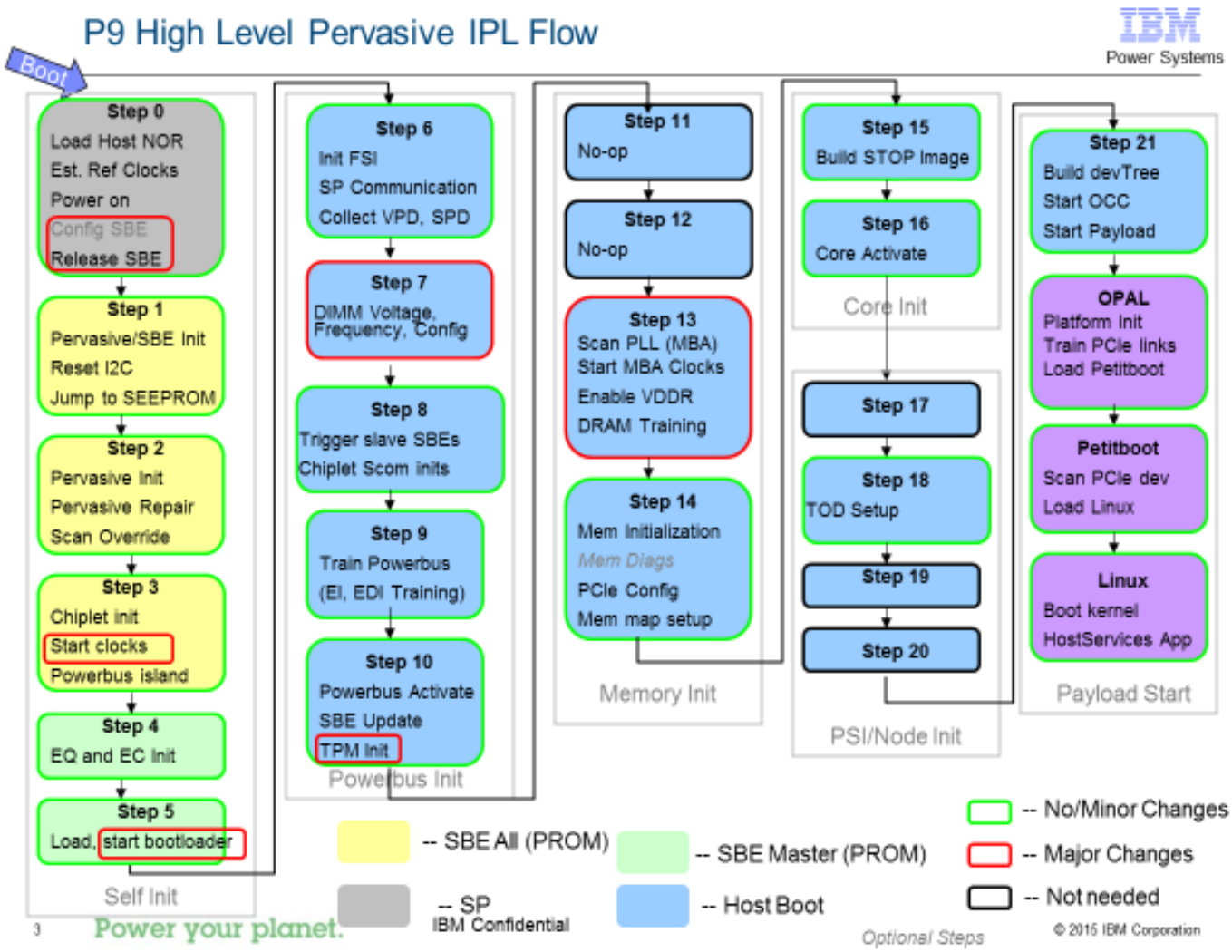
This version of the document will cover the following systems:

IBM FSP based system

Witherspoon – IBM BMC based system

Zaius – BMC based OpenPOWER system

The following diagram gives a high level overview of the IPL flow. The minute details are explained in the rest of the document.



1.2 Terminology

- **Centaur:** Memory buffer chip which optimizes memory bandwidth and usage
- **Cumulus:** P9 variant that only can access memory via attached Centaurs
- **Cronus:** Lab debug tool used to control and debug the IPL
- **DMI:** Dynamic memory Interface bus
- **IPL** : Inital Program Load = Boot process. Covers time between power on and running they hypervisor
- **HCODE: Hardware code** – Refers to the code the EX (EC, EQ) units between the running (full power), stop states (core power saved, cache active), and core and caches powered off states
- **Hostboot**: FW that initializes the memory and powerbus
- **HB Runtime Services (HBRT):** Portion of Host Boot that remains resident during hypervisor execution and provides PRD functionality for CE
- **istep** : IPL Step defined by ecmd interface
- **Nimbus:** P9 variant that has direct attached memory
- **Normal-mode** : IPL that includes minimal diagnostics, focused on functional requirements only

- **Maintenance-mode** : IPL that includes extensive diagnostics to test the hardware
- **OCC:** On Chip Controller – PPC 405 processor that controls the power management per chip
- **PCB:** Pervasive Control Bus – internal processor bus that provides an out of band communication layer between the internal logic within the chip
- **PNOR:** P9 Processor NOR chip. NOR flash device where all firmware, including hostboot firmware, is stored and
- from which it is loaded. It is attached to the master processor through an LPC → SPI bus connection. Called PNOR to distinguish from other NOR chips in the system
- **SBE:** Self Boot Engine – A version of a PORE within each P8 chip which is used to do some basic initialization to each chip and to load and start the Hostboot firmware

1.3 IPL Types

IPL Type	SP Power	CEC Standby Power	CEC Logic Power	Mainstore Contents	Applicable Platform
Standby POR	Off -> On	Off -> On	Off	Off	Pegasus
Cold IPL	On	On	Off -> On	Cleared	Pegasus
Warm IPL or Warm Re-IPL	On	On	On	Cleared	Pegasus

1.4 Nomenclature/Conventions

- Items in bold-italics represent deliveries from the hardware team, either procedures or engineering data files. Ex: ***p8_cfaminit.C***
- Steps in italics are software only and do not interact with the hardware at all.
- istep commands in black are performed by the Service Processor (SP)
- istep commands in blue are performed by the Self Boot Engine (SBE)
- istep commands in green are performed by the host code.
    - Note for multi-drawer or multi-Hostboot instances all commands are issued in parallel to all instances, except where otherwise noted
- istep commands in black are performed by the attached service processor

2 Service Processor Power On to Standby

2.1 FSP Based

This flow starts as soon as the power supplies are plugged in and ends when the SP reaches the standby state.

1. Base Wiring
    a) C_FSI_IN_ENA – tied to 0b1 for FSP to indicate that FSI is driven by FSI HW clock
    b) C_JTAG_TMS pin – tied to 0b0 to prevent autostart and allow BMC to select SBE/PNOR side (scratch regs)

2. Apply AC Power
    a) Standby regulators power on from 12VCS
    b) Reset generator starts 200ms after Standby Pgoods high
    c) Standby reset feeds into FSP/DPSS/APPS. (any intelligent device). FSP/APSS running. DPSS in reset if FSP present. If no FSP, then DPSS starts...Ckip steps 4-5.
    d) CFAM_RESET_B on all CFAMs (P9 and Centaur). Must be at least 100ms after VStandby
3. APSS loads itself from internal flash. We can update the APSS load in the lab from an internal connector. APSS is not updateable from FSP.
4. FSP starts running from SPI Flash at address 0.
5. FSP signs the DPSS load. There is only 1 – no golden image needed.
    a) Where the DPSS gets its load from the FSP
        - Held in FSP flash and clocked in
    b) DPSS load is updateable from the FSP, but not from the host
6. FSP releases DPSS reset.
7. DPSS begins running
    a) PGOOD Reset engine holds PGOODs off
    b) Registers initialized to default safe values (e.g. fans set to high speed).
    c) Waits for “GO” from FSP or external hardware entity to start power sequencing (DIO)
    d) From FSP, DPSS can be controlled to set LEDs, override fan speed (for fans on SB, if any),
8. FSP Machine type VPD read
9. FSI Clocks start to P9/Centaur
10. FSI break command sent to both slaves
11. FSI Arbitration is performed between Redundant FSPs

12. Read FSI Config Space
    a) Builds the device driver structure
    b) Device drivers “init”/clear engines – issues engine resets
13. Read all VPD (includes DRAM spd)
14. Start power process
15. Start HWServer process. At this point HWServer must build the object model based on VPD and chip IDEC
    a) istep BuildHwModel
        - This istep will build the HW model
        - HWServer will read the IDEC from the shift engine and the scom engine
        - During this phase (in non Reset-Reload cases) the primary FSP HWServer will force ownership of the LPC2SPI local bus for PNOR access
16. System now at SP Standby

At the end of this flow the SP and DPSS chip are powered on and viable. For the purposes of this discussion SP Standby is just enough information to power on the CEC logic. SP has **not** issued the “GO” bit yet.

If the P9 and Centaur chip's CFAM portion is powered on and has FSI clocks.

2.2 BMC Based

1. Base System wiring
    a) C_FSI_IN_ENA – tied to 0b1 for BMC to indicate that FSI is driven by ref clock
    b) C_JTAG_TMS pin – tied to 0b0 to prevent autostart and allow BMC to select SBE/PNOR side (scratch regs)
        - If want autostart without BMC control, then tie to 0b1
2. Apply AC Power
    a) Standby regulators power on from 12VCS
    b) Reset generator starts 200ms after Standby Pgoods high
    c) present. If no FSP, then DPSS starts...Ckip steps 4-5.
3. APSS loads itself from internal flash. We can update the APSS load in the lab from an internal connector. APSS is not updateable from FSP.
4. BMC starts running from SPI Flash at address 0.
5. BMC **option** to read VPD at standby voltage

a) Ref clock enabled and toggling (based on VSB power domain)

b) BMC would have toggle CFAM_RESET_B

c) BMC can then use SoftFSI to read VPD

- FSI Clocks start to P9/Centaur
- FSI break command sent to both slaves
- Read FSI Config Space
  - Builds the device driver structure
  - Device drivers “init”/clear engines – issues engine resets
- Read all VPD

6. BMC to power on:

a) Enable VDN

b) Ref clock enabled and toggling (based on VDN)

c) Toggle CFAM_RESET_B

d) Use SoftFSI to set CPU config/control regs (mailbox scratch registers for SBE side selection/mfg flags)

e) Use Soft FSI to kick off Master processor chip

2.3 SPLess Based

1. Base System wiring

a) C_FSI_IN_ENA – tied to 0b0 to indicate that FSI is driven by ref clock

b) C_JTAG_TMS pin – tied to 0b1 to allow autostart

2. Apply AC Power

a) Standby regulators power on from 12VCS – all power rails come up except VDDR

b) Reset generator starts 200ms after Standby Pgoods high

c) APSS loads itself from internal flash.

d) Ref clock enabled and toggling

e) HW Toggle CFAM_RESET_B

f) IPL Starts

3 Cold IPL

This flow covers the steps from FSP/BMC Standby through the initial handshake with PHYP/OPAL.

0 Step 0

```
0.1  poweron    : Power on system
0.2  startipl              : Start IPL on SP
```

- On warm re-ipl this is the entry point to the IPL flow
- Gets SP into a state ready to IPL the CEC

```
0.6  set_ref_clock (no-op on BMC)
```

a p9_setup_clock_term.C

- Setup the clock termination correctly for system/chip type
- Since this is the first procedure run against the chips it also clears the GP write protect
- Chip reference clocks start when their voltage rails come up, this step allows for the reference clock frequencies to be adjusted. Chip (Processor, Memory), PCIe, TOD (16Mhz)
  - For low end systems this is done via local I2C commands to the reference clock chip.

```
0.7  proc_clock_test (no-op on BMC)
```

a p9_select_clock_mux.C

- Select internal clock mux to drive the memory clocks off of
- Flips all bits needed for clock routing (processor only), centaur is done later in p9_cen_ref_clk_enable.C

b p9_clock_test.C

- Test to see if the ref clock is valid. If not switch to redundant clock or terminate IPL
- This is run prior to switching the frequency. It is intended to just see if the processor/memory are getting valid reference clocks
- NOTE: centaur doesn't have any clock logic to check for valid reference clocks, thus no procedure

```
0.8  proc_prep_ipl (no-op on BMC)
```

b p9_set_fsi_gp_shadow.C

- Corollary in BMC based system is the CFAM_RESET
- Done for all boots – some settings will change based on system type and IPL type

- Set the GP bits to default state
- Needs to take into account to not change values set up in p9_set_clock_term.C procedure

0.11 proc_select_boot_master

a p9_select_boot_master.C

- This HWP is misnamed due to historical reason, the actual selection of the master SBE is done in ***p9_setup_sbe_config.C***
- This HWP selects which Redundant SEEPROM to use
  - This must be set only for the master processor (HB will set later for slaves) depending on current IPL (normal or SBE update directed by Hostboot)
-

0.13 sbe_config_update

- On BMC systems this is done via direct writes to mbox scratch regs

b p9_setup_sbe_config.C

- See istep 2.2 for details of scratch registers and ATTR mappings
- This includes the Master/slave indication (for FSP/BMC it always sets master)
- Take the FSP/Cronus/hostboot FAPI2 ATTR and write them to the mbox scratch registers
- Data shuffling of the ATTR into an extremely compact form
- In manufacturing mode the SP may be required to update the entire seeproom image via xip_customize. See istep TBD for details
  - Note that the ring override from /nfs/ should be applied during the xip_customize flow if directly updating the SBE
  - Note to take into account the dead space between the 64KB SEEPROM images for SBE ECC

0.14 sbe_start

- Grant the LPC2SPI FSI bus to the LPC bus so the SBE and Hostboot can access the PNOR
- Done on all warm/cold IPLs under SP control.

b p9_start_cbs.C

- Set a bit to trigger the CBS on the P9 master chips. Located in FSI GP region
- CBS applies GP shadows to GP regs, causes endpoint resets
- The CBS will scan0 flush of pervasive, start clocks
- For MPIPL the CBS is not used and FSP directly triggers the SBE

1 Step 1 – Self Boot Engine OTPROM and PIBMEM

Note: release of FSI Go bit triggers SBE executing from OTPROM

a. Processor/Mem and FSI reference clocks are stable

b. SBE is reset to state that fetches directly from OTPROM (it is on the PIB)

- SBE instructions are parity protected, but SBE instruction parity is turned off (per VBU to make it easier for assembler compilers), but OTPROM is ECC protected
- SEEPROM, PNOR SBE, and partition NVRAM are ECC protected
- Hostboot, PHYP are CRC protected
- OTPROM is ECC protected
  - ECC checking is ON by default, scom bit to turn off ECC
  - Mechanism to stop SBE prior to any instructions issued is to use the FSI GP bit
- SEEPROM is ECC protected,
- Scan chains must be ECC protected in SEEPROM, ECC protected in OTPROM/PNOR

c. The following steps are done by the CBS. This happens regardless of the TMS line holding the SBE engine from fetching. The CFAM_RESET or FSI GP bit triggers this.

- Apply the root_ctrl shadow registers to the effective root_ctrl registers
- Init TP chiplet – NOTE that all iVRMs are in bypass (PgP only) (done by clock controller)
- Start TP Vital – TP mesh clocks (done by clock controller)
- Scan 0 flush – SBE can't scan PRV PIB or PCB regions as it is part of pervasive itself
  - Covers PCB and TP vital
  - This clears all security bits in the OTPROM controller AND the SDB bit in the mailbox
- Release tholds for TP and PCB – **running on the ref clock**
  - Pervasive clocks will be started by Clock Controller Logic; SBE itself receives these clocks and therefore can't run before clocks are running
  - Vital and PCB, not tholds to rest of TP
- PIB Bus operational now
- CBS triggers SBE start to fetch instructions

- SBE will not execute if external pin TE (Module/Wafer Manufacturing Test Enable) =0b1

1.1 proc_sbe_enable_seeprom :F,C - Select SEEPROM address

a This istep is not controllable by FW – once the CBS starts the boot sequencer the SBE will automatically execute this istep. It is listed as an istep for documentation, but cannot be manually controlled via istep.

b p9_sbe_enable_seeprom.C (no param) –

- ♦ Entrance into this procedure is via SBE Reset (hard) or CBS.
- ♦ Hard reset – triggered by SP (and potentially DTRM) without using the CBS
- ♦ CBS – runs scan 0 flush and clock start of PIB and NET domain (cleans up security latches) issues hard reset to SBE
- ♦ This HWP is not FAPI2 based:
  - It runs directly in OTPROM and cannot use attributes
  - It is burnt into the chips OTPROM during manufacture
- ♦ Running out of the OTPROM
- ♦ Select which redundant SEEPROM to use based on MBOX Control bit
  - 0b0 – use default SEEPROM (bit 17 of Self Boot Control/Status Register)
  - 0b1 – use alternate SEEPROM (bit 17 of Self Boot Control/Status Register)
- ♦ Resets the I2C bus
- ♦ If scratch reg is set then it uses I2C speed from scratch, else uses default burned into OTPROM at MFG
- ♦ Check that SEEPROM is accessible and image is valid (XIP header magic check)
- ♦ Then branch to SEEPROM location –
  - Magic number to address 0 (SBE) and jump point at address 0x4
  - Physically on the SEEPROM this will be 0x0

2 Step 2 Self Boot Engine – Pervasive Chiplet Setup

When the SBE first jumps to the SEEPROM it will jump to a routine delivered by FW to potentially collect FFDC based on the reset type. Then it will move to istep 2.1.

2.1 proc_sbe_ld_image :F,C - Load PIBMEM image

a This istep is not controllable by FW – once the CBS starts the boot sequencer the SBE will automatically execute this istep. It is listed as an istep for documentation, but cannot be manually controlled via istep.

- ♦ Not a FAPI HWP, instead raw C
  - RAW one that executes on the SBE – not against FAPI (OTPROM direct content)
  - Cannot use attributes
- ♦ Turn on SBE internal RISC trace via the SBE internal trace configuration register
- ♦ Performs PIBMEM repairs (via load/stores to PIB – aka scoms) only on start vector 0 (start vector 1 is used for warm resets and PIBMEM has already been setup and contains FFDC from/for the reset)
  - Data in pibmem is valid as long as previous steps did not go through scan 0 flush/clock start of PIB domain (CBS start does the scan0 flush)
  - SBE must always treat existing data in PIBMEM as FFDC only and always reload instructions
  - PIBMEM repairs are not required if the SBE is not being used (ie boot via FSI2PIB path)
    - For DFT if the PIBMEM repairs are needed, DFT is responsible for loading
- ♦ Loads the pib attached memory image from the SEEPROM This image contains various utilities used throughout the SBE IPL:
  - Kernel
  - Base Utilities
  - SBE fixed data section (aka ATTR) into PIBMEM
- ♦ Branch into SBE kernel, start executing Kernel
  - Enter control loop
  - After this point FW Control loop is in charge of loading/unloading chip ops and calling future HWP
  - SBE code checks the scratch registers to determine if in istep mode, if so then it enters istep mode and then waits for data on the FIFO. Otherwise it continues to boot automatically
    - If in non step mode, SBE will only honor FIFO operation to query IPL status/collect FFDC until it completes istep 5 or has an error
- ♦ All operations to the SBE are atomic from the SP perspective
- ♦ All power to the chip is on except
  - Quads
  - PHYs are all powered down

2.2 proc_sbe_attr_setup : F,C -Read scratch regs, update ATTR

a p9_sbe_attr_setup.C (chip target) FAPI2::ReturnCode

- If and only if scratch registers are non-zero, HWP will read the contents of the scratch registers and call FAPI2 APIs to set the values into the corresponding SBE platform ATTR values (Mbox reg contents PIBM ATTR)
  - Scratch 7, byte 0 is a bit field that indicates validity of the other mailbox register
- In the case where HW scratch registers are zero – the values represented by the scratch registers need to be in a fixed location (ECC aligned) of the SEEPROM Image (SEEPROM contents PIBMEM ATTR Mbox scratch regs)
  - SEEPROM image ATTR is the master, mailbox is just the overrides
  - Fixed location for the ATTR and all mbox ATTRs are at the front and non moveable (can extend, but not move)
  - In this case (scratch 7, byte 0 valid bit == 0) then data in the ATTR tank data in the SBE needs to be pushed back into the HW mailbox scratch reg for Hostboot to consume
- Hostboot will need the information in the scratch registers as well (for the slave chips, etc)
- Check the state of the SAB (Security Access Bit)
  - If SBE image has ATTR_SECURITY_MODE == 0b1, then leave SAB bit as is
  - Else ATTR_SECURITY_MODE == 0b0, then clear the SAB bit
    - ATTR_SECUIRTY_MODE may only be 0b0 with imprint keys
  - Move state of SAB into ATTR_SECURITY_ENABLE
    - Rest of the SBE code to apply security restrictions based on ATTR_SECURITY_ENABLE
- Mailbox scratch 1 (CFAM 2838, SCOM 0x50038) - FW functional EQ/EC
  - This register gives FW additional control over functional EQ/EC that the SBE can consider for bootable cores. It is applied on top of the manufacturing partial good VPD
  - Byte 0 EQ Gard records. Each bit position corresponds to chiplet (starting at chiplet 0x10 - 0x15) == ATTR_EQ_GARD (where 0x10 is bit 0 of byte 0)
    - EX functional is not explicitly represented
    - SBE can infer which EX (1/2 of EQ) are intended to be used based on the EC gard records
    - Do NOT need to support victim caches
  - Bytes 1-3 are EC Gard records. Each bit position corresponds to EC chiplet (starting at chiplet 0x20-0x37) == ATTR_EC_GARD (where 0x20 is bit 0 of byte 1)
  - This also information also need to go into the CME image

- If the bit is on then the part is non functional

♦ Mailbox scratch 2 (CFAM 2839, SCOM 0x50039) – SBE I2C Bus speed based, ref clock

- Bytes 0,1 are ref clock I2C bus divider consumed by code running out of OTPROM, no ATTR needed as it is directly read. ATTR is ATTR_I2C_BUS_DIV_REF (for image customization)
- Bytes 2
  - Bits 16:19 – ATTR_NDL_MESHCTRL_SETUP – Control NDL training, MeshCtrl setup
  - Bits 20-23 - Reserved
- Byte 3 is open

♦ Mailbox scratch 3 (CFAM 283A, SCOM 0x5003A) – FW Mode/Control flags

- The HWP does not need to do anything with this scratch register as it is SBE FW control flags. These will be stored as ATTR_BOOT_FLAGS in the ATTR tank (and by the setup mbox HWP). The SBE FW will check the valid bit and use the mbox scratch register, else it will use the value from the ATTR tank. The SBE FW will use these values prior to this HWP being run. Note that this is only used by the setup_sbe_config to push the data into the mailbox register when running on an FSP (not consumed by FAPI on SBE or as part of SBE SEEPROM customization).
  - Bit 0 indicates istep IPL (0b1) (Used by SBE, HB – FW ISTEP_MODE)
  - Bit 1 indicates that SBE should go directly to runtime functionality (0b1)
  - Bit 2 is reserved for HB usage for the SBE to indicate an MPIPL to Hostboot. It is always 0 in the ATTR tank and is dynamically set by the SBE at the same time the SBE sets the ATTR_MPIPL_MODE ATTR (Used by HB, set by SBE. SBE uses S0/S1 interrupt)
  - Bit 3 in this register is used to indicate FSPless (0b0), otherwise FSP attached (0b1)
  - Bit 4 -- Reserved
  - Bit 5 in this register indicates that the SBE should **not** send back internal FFDC on any ChipOp failure response
  - Bit 6 – disable security. SBE is configured to only honor this request if and only if during the update process it was signed with a secure header flag that permits it. Hostboot checks the secure header flag, signing server is responsible for never setting secure header flag with production keys

♦ Mailbox scratch 4 (CFAM 283B, SCOM 0x5003B) - Boot frequency

- Byte 0,1 -- EQ boot frequency multiplier == ATTR_BOOT_FREQ_MULT
  - Greg to provide algorithm
- Bit 16 – ATTR_CP_FILTER_BYPASS – force CP filter PLL into bypass

- Bit 17 -- ATTR_SS_FILTER_BYPASS – force SS filter PLL into bypass
- Bit 18 -- ATTR_IO_FILTER_BYPASS – force IO filter PLL into bypass
- Bit 19 -- ATTR_DPLL_BYPASS – force DPLL into bypass
- Bit 20 -- ATTR_NEST_MEM_X_O_PCIE_BYPASS– force nest PLL into bypass
- Bit 21 – ATTR_OBUS_RATIO_VALUE_BIT – Holds OBUS ratio value. 0b0 == normal speed, 0b1 == half speed
- Bit 22:23 -- Reserved
- Byte 3 -- Nest PLL bucket selection == ATTR_NEST_PLL_BUCKET
  - The PLL bucket number is an integer enum, with the actual frequency defined within the bucket
  - Where the PLL bucket contains a simple structure of the VDN setting, the Nest I2C divider setting, and then PLL ring, target nest frequency value in Khz (ie what system is targeted at, not necessarily the margin bias)
    - Supported buckets: 1600Mhz, 1866Mhz, 2000Mhz, 2133Mhz, 2400Mhz

♦ Mailbox scratch 5 (CFAM 283C, SCOM 0x5003C) – HWP Control Flags

- Bit 0 -- cache contained IPL (0b1), ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED
- Bit 1 -- SBE should init all cores (0b1), ATTR_SYS_FORCE_ALL_CORES == TRUE
- Bit 2 – HWP/Init “risk level” enabled (b1) – ATTR_RISK_LEVEL == 0x1
  - Note this is also used by Hostboot to pass to HB driven HWP
- Bit 3 – Boot loader HWP flag to not place 12K exception vectors. This flag is only applicable when security is disabled (ATTR_DISABLE_HBBL_VECTORS == 0x1)
- Bit 4 – Memory synchronous mode (0b1), ATTR_MC_SYNC_MODE == 0x1
- Bit 5 – Slow PCI reference clock (Nimbus DD1.0 only). 0b1 == ENUM_ATTR_DD1_SLOW_PCI_REF_CLOCK_NORMAL (100Mhz), 0b0 == ENUM_ATTR_DD1_SLOW_PCI_REF_CLOCK_SLOW (94Mhz).
- Bit 6:11 – Reserved/Open
- Bit 12:31 – Debug control for clock mux settings (20 bits), ATTR_CLOCK_PLL_MUX

♦ Mailbox scratch 6 (CFAM 283D, SCOM 0x5003D) – Master/Slave, node/chip selection

- Bit 23 – indicates if the chip is in group pump mode (ATTR_PROC_FABRIC_PUMP_MODE)
- Bit 24 – indicates Hostboot slave bit (ie not master), 0b0 == master, 0b1 == slave (ATTR_PROC_SBE_MASTER_CHIP has inverse polarity – ie a 0b1 when master, 0b0 when slave)
  - If set as slave then this overrides the external C4 indicating master/slave

- If set as master then use the external C4 as indication of master/slave
- The default SBE image will always have bit 24 indicating master (0b0), which will allow the board C4 pin to control master/slave
- For systems where the SP is intended to select master/slave, all module C4 pins must be tied low (indicating master) so that bit 24 will allow the SP to control master slave selection

- Bit 25 -- Reserved
- Bits 26:28 indicate the node position in FSP based systems (unused in Spless systems) ATTR_PROC_FABRIC_GROUP_ID
- Bits 29:31 indicate the chip position (ATTR_PROC_FABRIC_CHIP_ID)

♦ Mailbox scratch 7 (CFAM 283E, SCOM 0x5003E) – DRTM Payload address in MB

- Entire register used to indicate location of DRTM payload on MB boundary
- Only valid during DRTM execution

♦ Mailbox scratch 8 (CFAM 283F, SCOM 0x5003F)

- Byte 0 – each bit in here indicates validity of the same numbered scratch reg (bit 0 scratch 0)
  - Bit 0 -- (CFAM 2838, SCOM 0x50038) - FW functional EQ/EC valid
  - Bit 1 -- (CFAM 2839, SCOM 0x50039) - SBE I2C Bus speed based, ref clock valid
  - Bit 2 -- (CFAM 283A, SCOM 0x5003A) - FW Mode/Control flags valid
  - Bit 3 -- (CFAM 283B, SCOM 0x5003B) - Boot frequency valid
  - Bit 4 -- (CFAM 283C, SCOM 0x5003C) - HWP Control Flags valid
  - Bit 5 -- (CFAM 283D, SCOM 0x5003D) - Master/Slave, node/chip selection valid
  - Bit 6 -- (CFAM 283E, SCOM 0x5003E) – DRTM Payload address in MB valid
  - Bit 7 -- (CFAM 283F, SCOM 0x5003F) – bytes 1,2,3 (if used) valid
- This is used to know if the data should be updated from scratch to attributes

2.3 proc_sbe_tp_chiplet_init1 :F,C,D TP Chiplet Init

a p9_sbe_tp_chiplet_init1.C (chip target) FAPI2::ReturnCode

♦ Releases the Pervasive Control Bus (PCB) reset

♦ Sets TP chiplet enable

♦ Drops pervasive chiplet fence

2.4 proc_sbe_tp_gptr_time_initf :F,C,D – Init Perv GPTR/Time

a p9_sbe_tp_gptr_time_initf.C

- Scan init the GPTR and Time rings for the Pervasive chiplet

2.5 proc_sbe_dft_probe_setup_1 :D, - Setup DFT probe points

a p9_sbe_dft_probe_setup_1.C (chip target) FAPI2::ReturnCode

- Only run in DFT mode, no-op in normal Cronus/SBE (istep stub for common numbering)
- DFT mode is controlled with IPL option within Cronus

2.6 proc_sbe_npll_initf :F,C,D - Program Powerbus PLL

a p9_sbe_npll_initf.C (chip target) FAPI2::ReturnCode

- Apply the Nest PLL ring
- Nest PLL ring is picked off of scratch reg bucket selection
  - Must run at system frequency
  - Consists of compressed scan ring in SEEPROM. There are 4 buckets (1.8, 2.0, 2.13, 2.4)This image is set via p9_xip_customize based off of the system Nest/Xbus PLL setting. There are two potential images for each bucket:
    - Normal
    - Override – this is an image that can be selected to override to a custom PLL setting for the nest
    - When SECURITY_ENABLE is set, scan overrides can only come from a known good scan ring whitelist (PLLs)
- Obus, PCIe, and MC PLLs are not set (still running in bypass)

2.7 proc_sbe_npll_setup : F,C,D - Nest PLL setup

a p9_sbe_npll_setup.C (chip target) FAPI2::ReturnCode

- Clocking: set nest sector buffer strength, pulse mode and pulse mode enable (attribute dependency Nimbus/Cumulus)
- Clocking: Apply Nest Progdly (dependency to VPD #MK) setting
- Clocking: enable Nest Progdly (set nest progdly bypass to zero)
- Get Nest running, check PLL, makes use of a glitchless mux to switch

2.8 proc_sbe_tp_switch_gears : F,C,D - Update SBE I2C config

a p9_sbe_tp_switch gears.C (chip target) FAPI2::ReturnCode

- Calls procedure to update I2C bus speed in the PIBMEM

2.9 proc_sbe_clock_test : F,C,D – Check clocks

a ***Noop***

-

2.10 proc_sbe_tp_chiplet_reset : F,C,D – Reset TP Chiplet

a ***p9_sbe_tp_chiplet_reset.C (chip target) FAPI2::ReturnCode***

- Setup hang counter for PCB slaves/master

2.11 proc_sbe_tp_repr_initf : F,C,D - TP Chiplet Repair

a ***p9_sbe_tp_repr_initf.C (chip target) FAPI2::ReturnCode***

- Load Scan Repair for TP Chiplet

2.12 proc_sbe_tp_chiplet_init2 : F,C,D - TP Chiplet Repair

a ***p9_sbe_tp_chiplet_init2.C (chip target) FAPI2::ReturnCode***

- Scan 0 all rings on TP – including occ, perv. This excludes the PIB, PCB, Repair, Time, and GPTR rings (as this is where SBE is running from and were done by the Clock controller logic)

2.13 proc_sbe_setup_tp_abist g: D -- Hook for DFT to run abist on TP

a ***p9_sbe_tp_abist_setup.C (chip target) FAPI2::ReturnCode***

- Spot for DFT to insert non zero (ie true abist) patterns

2.14 proc_sbe_tp_arrayinit :F,C,D - TP Chiplet array init

a ***p9_sbe_tp_arrayinit.C (chip target) FAPI2::ReturnCode***

- Does not reinit PIBMEM
- Run arrayinit on TP chiplet (includes OCC)
  - After this all TP arrays are initialized (including OCC SRAM tank)
- Scan flush 0 to all TP expect TP Time, GPTR, Repair rings and PIB, and PCB regions

2.15 proc_sbe_tp_initf :F,C,D - TP Chiplet scan inits

a ***p9_sbe_tp_initf.C (chip target) FAPI2::ReturnCode***

- Apply scan overrides to TP Chiplet (includes OCC)

2.16 proc_sbe_dft_probe_setup_2 :D, - Setup DFT probe points

a ***p9_sbe_dft_probe_setup_2.C (chip target) FAPI2::ReturnCode***

- Only run in DFT mode, no-op in normal Cronus/SBE (stub istep left for common numbering)

2.17 proc_sbe_tp_chiplet_init3 :F,C,D - TP Chiplet Start clocks

a ***p9_sbe_tp_chiplet_init3.C (chip target) FAPI2::ReturnCode***

- Switches TP Chiplet OOB mux
- Resets PCB Master Interrupt register

- Drops pervasive and OCC2PIB fence
- Start clocks on perv region (all components of TP)
- Clear force_align in chiplet GP0
- Clear flushmode_inhibit in chiplet GP0
- Drop FSI fence so checkstop and interrupt conditions can flow – SBE has direct path, this is normal TP chiplet path
- Pervasive Trace arrays are now available
- Check for OSC switch clock errors after switching to Nest PLLs
- Theoretically can run the OCC at this point
- If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED
  - Tweak FIR Masks

3 Step 3 Self Boot Engine – Chiplet Setup

3.1 proc_sbe_chiplet_reset : F,C,D -Nest Chiplet Reset

a

b p9_sbe_chiplet_reset.C (chip target) FAPI2::ReturnCode

- Setup static multicast groups for all good chiplets excluding TP based on pervasive target functional state (not ATTR_PG state)
  - If ATTR_SYS_FORCE_ALL_CORES == true, then add all EQ/EC to the multicast groups
  - Otherwise the EQ/EC multicast will be added late in the IPL flow by proc_select_ex or in step 15 by Hostboot.
  - Step 3 can't use the multicast for all non-nest chiplets (ie EQ/EC)
- For all good chiplets including EQ/EC
  - Setup chiplet net control regs
    - Reset PCB Slave to default state
    - Set chiplet enable on all all good chiplets
- Clocking: setup chiplet sector buffer strength, pulse mode and pulse mode enable (attribute dependency Nimbus/Cumulus)
- Setup of hang counters including EQ/EC

- For all enabled good chiplets excluding EQ/EC
  - Start vital clocks and release endpoint reset
  - PCB Slave error register Reset

3.2 proc_sbe_gptr_time_initf: Init GPTR, Time rings for chiplets

a p9_sbe_gptr_time_initf.C

- Scan initalize all rings and initialize REPR on all enabled chiplets (except for TP, EP and EC)

3.3 proc_sbe_chiplet_pll_initf : PLL Initfile for X, O, PCIe, DMI, MCA

a p9_sbe_chiplet_pll_initf.C

- PLL rings are stored in SBE image
- Included tune bits, frequency
- Includes issuing the set pulse

3.4 proc_sbe_chiplet_pll_setup : Setup PLL for O, X, PCIe, DMI, MCA

a p9_sbe_chiplet_pll_setup.C

- Clocking: MC Chiplet only, Setup DCC and Progdlys
  - Progdlys (Nimbus two entries), dependency to VPD #MK field
  - Progdly (Cumulus one entry), dependency to VPD #MK field
  - DCC attribute dependency Nimbus/Cumulus
- Clocking: drop DCC and Progdly bypass signals
- Checks that the PLL locked
- Start the VAR OSCs / Config the TANK PLLs & lock
- In certain configs these chiplets are potentially not used
- Must run at system frequency
- If in async mode the MCA PLLs are locked to default PLL chain (mem PLL bucket for 2Ghz)
- Else if in sync mode then MCA PLLs are not enabled because the MCAs are driven from the nest PLLs

3.5 proc_sbe_repr_initf : F,C,D -Chiplet Repair

a p9_sbe_repr_initf.C (chip target) FAPI2::ReturnCode

- For all enabled chiplets
  - Scan 0 all rings on all enabled chiplets (except for TP)
  - Load Repair, Time and GPTR rings for all enabled chiplets

- All chip customization data is within the Repair and Time rings – array repair, DTS settings

3.6 proc_sbe_chiplet_init : F,C,D -Chiplet Init

a p9_sbe_chiplet_init.C (chip target) FAPI2::ReturnCode

- For all enabled chiplets
  - Scan 0 all rings (except time, repair, gptr) on all enabled chiplets

3.7 proc_sbe_abist_setup : D -- Hook for DFT to run abist

a p9_sbe_abist_setup.C (chip target) FAPI2::ReturnCode

- Stub for DFT – requirement is not to be compiled into real SBE/CME/GPE image – only an istep placeholder
- Spot for DFT to insert non zero (ie true abist) patterns

3.8 proc_sbe_arrayinit : Chiplet array init

a p9_sbe_arrayinit.C

- Run arrayinit on all enabled chiplets
- Scan flush 0 to all rings except GPTR, Time, Repair on all enabled chiplets

3.9 proc_sbe_lbist :D -- Hook for DFT to run lbist

a p9_sbe_lbist.C

- Stub for DFT – requirement is not to be compiled into real SBE/CME/GPE image – only an istep placeholder
- Run lbist on all enabled chiplets
- Scan flush 0 to all rings except GPTR, Time, Repair on all enabled chiplets

3.10 proc_sbe_tp_enable_ridi : Put Enable pervasive RIDI

a p9_sbe_tp_enable_ridi.C (chip target) FAPI2::ReturnCode

- Drop RI/DI for the AVS bus
- Drop RI/DI for TP logics

3.11 proc_sbe_setup_boot_frequency : Setup boot frequency

a p9_sbe_setup_boot_frequency.C

- Read core frequency ATTR and write to the Quad PPM

3.12 proc_sbe_initf : Apply any scan overrides

a p9_sbe_nest_initf.C

- Initfiles in procedure defined on VBU ENGD wiki
- Apply scan overrides to all enabled chiplets

- Generated via “traditional” initfile, but stored as compressed RS4 scan rings
- Spot to put all differences from scan flush 0
  - Intended only for config independent settings “patches”. Chip team goal is to flush to the correct state
  - Cannot contain system configuration differences, but can contain chip customization settings (ie DMI vs EDI personalization)
- Primary debug mechanism is to use Cronus/FSP putspy commandline to modify ring images directly in the chip (ie istep, then putspy).
  - Doesn’t cover core
  - Need to know when in the IPL you can perform the scan ring
  - Doesn’t cover system test (ie non script/interactive mode)
- Secondary mechanism is to build an RS4 overlay and have a mechanism/location for the SBE to pick-up various overlays and apply
  - Required for core
  - Mechanism to provide system test with patches
    - identify storage tank for overlays, RS4 is self-describing, put hook into SBE to walk rings and look for nest/MC chiplet overlays

3.13 proc_sbe_nest_startclocks : Start PB and nest clocs

a p9_sbe_nest_startclocks.C

- Drop fences and tholds on PB Chiplets
- Start nest chiplets with N3 as the master, rest as the slave
- Note that although the MCS logic is started (part of the Nest),
- If in async mode the MCA/ memory chiplets are not are started here.
- Else if in sync mode the MCA/memory chiplets are started here.
- In either case the MCA has the PLL/grid running, but not necessarily the functional clocks (in async mode)

3.14 proc_sbe_nest_enable_ridi : Enable nest RI/DI

a p9_sbe_nest_enable_ridi.C

- Drop RI/DI for nest -- LPC and PSI IOs

3.15 proc_sbe_io_initf : Apply inits to chipl IOs

a p9_sbe_io_initf.C

- Apply init file for chiplet IOs

3.16 proc_sbe_startclock_chiplets : Start clocks on O, X, PCIe

a p9_sbe_startclock_chiplets.C

- Start Xbus, Obus, PCIe clocks
- Start Mem chiplet if it is in synchronous mode
- Start clocks on configured chiplets

3.17 proc_sbe_scominit : SBE Nest scominits

a p9_sbe_scominit.C (processor chip)

- Apply any scom inits to nest chiplets
- If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED
  - Tweak FIR Masks

3.18 proc_sbe_lpc : Init the LPC master

a p9_sbe_lpc_init.C

- Requirement from the bootloader is that it only uses MMIOs to LPC master, not Xscom
- Perform scoms to setup LPC bus
  - Move the LPC clock to external input
- Pull the LPC unit out of reset
- Set LPC BAR – hardcoded like Xscom BAR

3.19 proc_sbe_fabricinit : Init fabric(PB) for island mode

a p9_sbe_fabricinit.C

- Send fabric command and check result
  - Chip will scan flush to SMP island mode
- This initializes PgP chip in “island” fabric mode and allows the core access to the PIB
- Pbus will flush to a state where all chiplets come up as good configured and disconnected – logic in powerbus respond to snoop with NULL response (traditional way of handling STOP)
  - In single chip mode Obus and Xbus, memory units come up fenced
- As chiplets come online then fabric must be “connected” to the chiplet
  - EX – controlled by winkle
  - Xbus, Abus – Hot add operation
  - memory units – nest facing MCS logic is in N1/N3, already initialized

- What about PCIe chiplets -- nest facing PCIe logic is in N2, already initialized
- Chiplets that are not used (deconfigured) are left in this state

3.20 proc_sbe_check_master : Determine if master chip

a At this point the SBE must use the internal bolt-on register to toggle TPM Reset line

b Determine if this is master SBE

- ♦ SBE FW checks bit 24 of the Scratch register (stored in ATTR) –
- ♦ if set then this is a slave chip, load /enable runtime chipOps
- ♦ else master and continue

3.21 proc_sbe_mcs_setup : Setup MCS to allow EX contained

- ♦ This step needs to be a no-op on MPIPL/DRTM flow

b p9_sbe_mcs_setup.C

- ♦ If
  - ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED this step is a no-op
- ♦ Else:
  - Open the MCS BAR to allow Hostboot to dcbz the contents of cache.
  - Also disable speculative pre-fetch to prevent PBA reads from triggering operations to MCS

3.22 proc_sbe_select_ex : Select Hostboot core

a p9_sbe_select_ex.C

- ♦ FW will have correctly set the target functional state(s). HWP uses functional states as master record (doesn't need to read PG data, gard, etc)
- ♦ If ATTR_SYS_FORCE_ALL_CORES is set
  - then force select to ALL
    - Multicast groups are already setup by istep 3.1
  - Else single "master core"
    - the first functional EC/EQ is the master core. Note that in this mode no EQ/ECs have been added to any multicast group before this point
    - need to add master EC to multicast group 0, 1, 3
    - need to add master EQ to multicast group 0, 4 (and EX to 5, 6 as needed)

- Write selected (single/all) EQ/Core mask into OCC complex
- This is the “master record“ of the enabled cores/quad in the system
- This is only for during the IPL (will be updated later in step 15)

4 Step 4 Self Boot Engine – EX Init

Note: Master chip (attached PNOR) inits EX unit for Hostboot execution image. Slave chips patiently wait.

- Issue isteps detailed in EQ and EC section
  - These are common to STOP images
  - Execution will return here afterwards
- Does NOT start instructions on core

Cache Initialization

This flow covers the steps that are used to initialize the Cache chiplet. Although it is inserted in the mainline IPL flow, it is executed both in the IPL (to bring up the cache associated with the HostBoot core) and for the STOP GPE execution used during run-time. After this flow is done, the flow (once appropriate core multicast groups are established) described in Core Initialization can be executed.

All cache initialization is done using Multicast Group 6. Therefore, the caches (i.e. cache chiplets) that are to participate need to have that group number set into one of the PCBS Multicast registers prior to invoking this flow. For IPL, the number of caches is dependent on the the number of cache chiplet specific sets of information that will fit into the SEEPROM (minimum: one but can be more); for STOP, this can be any combination of cache chiplets (subsetted by partial good and gard settings) as all good (from manufacturing) caches will have chiplet specific information (i.e. repair ring data) in the HOMER region.

4.1 proc_hcd_cache_poweron : Cache Chiplet Power-on

a p9_hcd_cache_poweron.C

- Command the cache PFET controller to power-on
- Check for valid power on completion
  - Polled Timeout: 100us
- For Nimbus DD1.0 only enable Vdd PFETS, do not enable Vcs PFETS – controlled by feature ATTR

4.2 proc_hcd_cache_chiplet_reset : Cache Chiplet Reset

a p9_hcd_cache_chiplet_reset.C

- Reset quad chiplet logic
- Clocking: setup cache sector buffer strength, pulse mode and pulsed mode enable values (attribute dependency Nimbus/Cumulus)
- Clocking: Drop glsmux async reset

- Scan0 flush entire cache chiplet

4.3 proc_hcd_cache_chiplet_l3_dcc_setup : Cache Chiplet DCC Setup

a p9_hcd_cache_chiplet_l3_dcc_setup.C

- Clocking: Setup L3 DCC (scan with setpulse, scan region = ANEP), attribute dependency Nimbus/Cumulus
- Clocking : drop L3 DCC bypass

4.4 proc_hcd_cache_gptr_time_initf : GPTR and Time for EX non core

a p9_hcd_cache_gptr_time_initf.C

- Initfiles in procedure defined on VBU ENGD wiki to produce #G VPD contents
- Check for the presence of core override GPTR ring from image (this is new for P9)
- if found, apply; if not, apply core GPTR from image
- Check for the presence of core override TIME ring from image;
- if found, apply; if not, apply core base TIME from image

4.5 proc_hcd_cache_dpll_initf : Quad DPLL Setup

a p9_hcd_cache_dpll_initf.C

- Initfiles in procedure defined on VBU ENGD wiki
- DPLL tune bits are not dependent on frequency
- Put DPLL into bypass
- Set DPLL syncmux sel
- Set clock controller scan ratio to 1:1 as this is done at refclk speeds
- Load the EX DPLL scan ring
- Set clock controller scan ratio to 8:1 for future scans

4.6 proc_hcd_cache_dpll_setup : Quad DPLL Setup

a p9_hcd_cache_dpll_setup.C

- Frequency is controlled by the Quad PPM
  - Actual frequency value for boot is stored into the Quad PPM by p9_hcd_setup_evid.C in istep 2
  - In real cache STOP exit, the frequency value is persistent
- Enable the DPLL in the correct mode
  - non-dynamic

- Slew rate established per DPLL team

- Take the cache glitchless mux out of reset
- Remove DPLL bypass
- Drop DPLL Tholds
- Check for DPLL lock
  - Timeout: 200us
- Switch cache glitchless mux to use the DPLL

4.7 proc_hcd_cache_dcc_skewadjust_setup : Quad DCC skew adjusts

a p9_hcd_cache_dcc_skewadjust_setup.C

- Start Clocks clock region = AN only
- Drop DCCs reset
- Setup 6 DCCs in parallel (commands over scan with setpulse, scan region = ANEP), dependency to VPD field #MK
- Drop DCCs bypass
- Additional DCC setup step (commands over scan with setpulse, scan region = ANEP)
- Drop SkewAdjust reset
- Setup Skewadjust (commands over scan with setpulse, scan region = ANEP), dependency to VPD field #??
- Drop SkewAdjust bypass
- Additional SkewAdjust setup step (commands over scan with setpulse, scan region = ANEP)

4.8 proc_hcd_cache_chiplet_init : EX Flush/Initialize

a p9_hcd_cache_chiplet_init.C

- Scan0 flush all configured chiplet rings except Vital, GPTR, TIME and DPLL

4.9 proc_hcd_cache_repair_initf : Repair ring for EX non core

a p9_hcd_cache_repair_initf.C

- This HWP is run serialized per EQ (most others are done in multicast)
- Load cache ring images from MVPD
  - These rings must contain ALL chip customization data. This includes the following: Repair Power headers, and DTS
  - Historically this was stored in MVPD keywords are #R, #G. Still stored in MVPD, but SBE image is

customized with rings for booting cores

4.10 proc_hcd_cache_arrayinit : EX Initialize arrays

a p9_hcd_cache_arrayinit.C

- Use ABIST engine to zero out all arrays
- Upon completion, scan0 flush all rings except Vital, Repair, GPTR, TIME and DPLL

4.11 proc_hcd_cache_abist : DFT hook for abist

a p9_hcd_cache_abistabist.C

- Stub for DFT – requirement is not to be compiled into real SBE/CME/GPE image – only an istep placeholder
- Upon completion, scan0 flush all rings except Vital, Repair, GPTR, TIME and DPLL

4.12 proc_hcd_cache_lbist : DFT hook for lbist

a p9_hcd_cache_lbist.C

- Stub for DFT – requirement is not to be compiled into real SBE/CME/GPE image – only an istep placeholder
  - Use LBIST engine to run tests
  - Upon completion, scan0 flush all rings except Vital, Repair, GPTR, TIME and DPLL

4.13 proc_hcd_cache_initf :EX (non core) scan init

a p9_hcd_cache_initf.C

- Initfiles in procedure defined on VBU ENGD wiki
- Call putring on EQ rings
  - Putring checks for the presence of cache FUNC override/cache contained/risk level/etc rings from image;
  - if found, apply; if not, apply cache base FUNC rings from image
- Note: FASTINIT ring (eg CMSK ring) is setup at this point to limit the stumps that participate in FUNC ring scanning (this is new for P9).
- Note: all caches that are in the Cache Multicast group will be initialized to the same values via multicast scans
  - Note that this is done 2X – once for even EX in EQ and once for odd EX in EQ

4.14 proc_hcd_cache_startclocks : Quad Clock Start

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p9_hcd_cache_startclocks.C

- Set (to be sure they are set under all conditions) core logical fences (new for P9)
- Drop pervasive thold
- Setup L3 EDRAM/LCO
- Drop pervasive fence
- Reset abst clock muxsel, sync muxsel
- Set fabric node/chip ID from the nest version
- Clear clock controller scan register before start
- Start arrays + nsl regions
- Start sl + refresh clock regions
- Check for clocks started
  - If not, error
- Clear force align
- Clear flush mode
- Drop the chiplet fence to allow PowerBus traffic

4.15 proc_hcd_cache_scominit : Cache SCOM Inits

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p9_hcd_cache_scominit.C

- Apply any SCOM initialization to the cache
- Setup L3 configuration mode (LCO)
- Configure Trace Stop on Xstop
- DTS Initialization sequence

4.16 proc_hcd_cache_scom_customize : Cache Customization SCOMs

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p9_hcd_cache_scomcust.C

- Dynamically built (and installed) routine that is inserted by the "XIP Customization" process. (New for P9)
- Dynamically built pointer where a NULL is checked before execution
- If NULL (a potential early value); return
- Else call the function at the pointer; pointer is filled in by XIP Customization

- Customization items:
  - Epsilon settings scan flush to super safe
    - Customize Epsilon settings for system config
  - LCO setup (chiplet specific)
    - FW setups up based victim caches
  - Powerbus (MCD) and L3 BAR settings

From this point on, all data added to the image is for run-time modifications for STOP

4.17 proc_hcd_cache_ras_runtime_scom : EX Runtime Scom Init

a p9_hcd_cache_ras_runtime_scom.C

- Not consumed by SBE (empty istep); SGPE only
- Run-time updates from Host based PRD, etc that are put on the core image by STOP API calls
- Dynamically built pointer where a NULL is checked before execution
- If NULL (the SBE case), return
- Up to three separate sections – normal scom, L2 repair, and L3 repair
- Else call the function at the pointer; pointer is filled in by STOP image build
  - Runtime FIR mask updates from PRD
  - L2/L3 repairs

4.18 proc_hcd_cache_occ_runtime_scom : EX OCC runtime SCOMS

a p9_hcd_cache_occ_runtime_scom.C

- Not consumed by SBE (empty istep); SGPE only
- Run-time updates from OCC code that are put here
- OCC FW sets up value in the TBD SCOM section
- Placeholder at this point

Note: this flow does NOT do anything with any of the cores attached to the caches that were just initialized. Also, this portion of the flow does also NOT initialize the CMEs in the cache chiplet as this cannot be done when this flow is run as a part of istep 4.

Core Initialization

This flow covers the steps that are used to initialize the Core chiplet. It is covered prior to the mainline IPL flow as it is a separate image that is executed both in the IPL (to bring up the HostBoot core) and for the CME STOP execution. The

running of this initialization flow REQUIRES the flow described in Cache Initialization to have been previously executed. In the Cache flow, the PCBS for the cores are endpoint reset but nothing behind the PCBS is affected (eg the EPS components as the power to these are not yet on).

4.19 proc_hcd_exit_mode : Determine which Cores to process

- Stub for SBE (empty istep)
- SGPE/CME have logic here to determine which cores should be acted upon

4.20 proc_hcd_core_pcb_arb : Core Chiplet PCB Arbitration

a p9_hcd_core_pcb_arb.C

- If CME, request PCB Mux.
  - Poll for PCB Mux grant
- Else (SBE)
  - Nop (as the CME is not running in bringing up the first Core)

4.21 proc_hcd_core_poweron : Core Chiplet Power-on

a p9_hcd_core_poweron.C

- Command the core PFET controller to power-on
- Check for valid power on completion
  - Polled Timeout: 100us

4.22 proc_hcd_core_chiplet_reset : Core Chiplet Reset

a p9_hcd_core_chiplet_reset.C

- Reset chiplet logic
- Clocking: setup core sector buffer strength, pulse mode and pulsed mode enable values,), attribute dependency Nimbus/Cumulus
- Clocking: Drop glsmux async reset
- Scan0 flush entire core chiplet

4.23 proc_hcd_core_gptr_time_initf : Load Core GPTR and Time rings

a p9_hcd_core_gptr_time_initf.C

- Initfiles in procedure defined on VBU ENGD wiki to produce #G VPD contents
- GPTR is common between cores (ie multicast / PCB muxing)
- Check for the presence of core override GPTR ring from image (this is new for P9)

- if found, apply; if not, apply core GPTR from image

♦ Check for the presence of core override TIME ring from image;

- if found, apply; if not, apply core base TIME from image

4.24 proc_hcd_core_chiplet_init : Core Flush/Initialize

a p9_hcd_core_chiplet_init.C

♦ Switch the core glitchless mux to allow DPLL clocks on the clock grid

♦ Clocking: setup controls based on DPLL frequency

♦ Clocking: assert PM sync_enable (4x core, 2 x L2), DCCs and SkewAdjust starts aligning clocks

♦ Scan0 flush all chiplet rings except VITAL, GPTR and TIME

4.25 proc_hcd_core_repair_initf: Load Repair ring for core

a p9_hcd_core_repair_initf.C

♦ This step is run individually per core (serialized)

♦ Load core ring images from that came from MVPD into the image

- These rings must contain ALL chip customization data. This includes the following: Array Repair and DTS calibration settings

- Historically this was stored in MVPD keywords are #R, #G. Still stored in MVPD, but SBE image is customized with rings for booting cores at build time

4.26 proc_hcd_core_arrayinit : Core Initialize arrays

a p9_hcd_core_arrayinit.C

♦ Use ABIST engine to zero out all arrays

♦ Upon completion, scan0 flush all rings except Vital, Repair, GPTR, and TIME

4.27 proc_hcd_core_abist : DFT hook for abist

a p9_hcd_core_abist.C

♦ Stub for DFT – requirement is not to be compiled into real SBE/CME/GPE image – only an istep placeholder

♦ Upon completion, scan0 flush all rings except Vital, Repair, GPTR, TIME and DPLL

4.28 proc_hcd_core_lbist : DFT hook for lbist

a p9_hcd_core_lbist.C

♦ Stub for DFT – requirement is not to be compiled into real SBE/CME/GPE image – only an istep placeholder

♦ Use LBIST engine to run tests

- Upon completion, scan0 flush all rings except Vital, Repair, GPTR, TIME and DPLL

4.29 proc_hcd_core_initf :Core scan init

a p9_hcd_core_initf.C

- Initfiles in procedure defined on VBU ENGD wiki
- Check for the presence of core FUNC override rings from image;
- if found, apply; if not, apply core base FUNC rings from image
- Note: FASTINIT ring (eg CMSK ring) is setup at this point to limit the stumps that participate in FUNC ring scanning (this is new for P9).
- Note : if in fused mode, both core rings will be initialized to the same values via multicast scans

4.30 proc_hcd_core_startclocks : Core Clock Start

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p9_hcd_core_startclocks.C

- Drop pervasive thold
- Drop pervasive fence
- Reset abst clock muxsel, sync muxsel
- Clear clock controller scan register before start
- Start arrays + nsl regions
- Start sl + refresh clock regions
- Check for clocks started
  - If not, error
- Clear force align
- Drop the core to cache logical fence

4.31 proc_hcd_core_scominit : Core SCOM Inits

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p9_hcd_core_scominit.C

- Apply any coded SCOM initialization to core

4.32 proc_hcd_core_scom_customize :Core Customization SCOMS

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p9_hcd_core_scomcust.C

- Dynamically built (and installed) routine that is inserted by the "XIP Customization" process. (New for P9)
- Dynamically built pointer where a NULL is checked before execution
- If NULL (a potential early value); return
- Else call the function at the pointer; pointer is filled in by XIP Customization

From this point on, all data added to the image is for run-time modifications for STOP

4.33 proc_hcd_core_ras_runtime_scom : EX Runtime Scom Init

a p9_hcd_core_ras_runtime_scom.C

- Not consumed by SBE (istep is placeholder); CME only
- Run-time updates from Host based PRD, etc that are put on the core image by STOP API calls
- Dynamically built pointer where a NULL is checked before execution
- If NULL (the SBE case), return

Else call the function at the pointer; pointer is filled in by STOP image build

4.34 proc_hcd_core_occ_runtime_scom : Core OCC runtime SCOMS

a p9_hcd_core_occ_runtime_scom.C

- Not consumed by SBE (istep placeholder); CME only
- Run-time updates from OCC code that are put here
- OCC FW sets up value in the TBD SCOM section. This was not leverage in P8 with the demise of CPMs
- Placeholder at this point

Note: for STOP image wakeup (eg istep 4), there is no explicit instruction start here. In Cronus mode this was left up to the user, in FW the instruction control is managed in step 5. However for true STOP usage the CME blocks interrupts to core, puts HRMOR in place to point the Core to HOMER and issues an SRESET to all threads. The threads goes into HOMER, and do SPR restoration. Then slave threads end at STOP level 15, master thread waits for slaves to complete (reach STOP15) then switches to "real" HRMOR and other SPRs and then enters STOP15. When CME sees all threads at STOP15, it unblocks interrupts, which allows normal execution to commence.

5 Step 5 Self Boot Engine – Load Hostboot

Note: Master chip (attached PNOR) loads Host boot image. Slave chips patiently wait. Master SBE fetches Hostboot code from PNOR and places in target EX

5.1 proc_sbe_load_bootloader

a p9_pm_ocb_indir_setup_linear.C

- Setup OCB channel 3 to linear mode

b p9_sbe_load_bootloader.C

- Setup PBA to target specific cache (L3 tank)
- SBE fetches bootloader, security algorithm, and hash of HW public keys from SEEPROM
  - SEEPROM Image is ECC protected
  - Design is still in discussion, but each of the items above are independent (ie the key hash and bootload, security code will want to be updated from HB independently). They are NOT part of SBE xip customize image (but SBE knows how to find)
- Places bootloader at specific address
  - 0x 08200000 + 12KB (HRMOR of 130MB, ie 2MB into 10MB cache) – tentative bootloader address
- SBE fetches signature validation code from SEEPROM, places at specific address
- SBE fetches hash of HW public keys from SEEPROM, places at specific address
- SBE creates POWER interrupt table (12K)
  - Done by SBE code because we don't want to waste 12K of SEEPROM space
  - Current idea is a branch absolute to 12KB
- SBE does not open an unsecure memory window -- Host has to indicate to SBE what the unsecure memory window is
  - In other words SBE Chip Ops won't let PBA/ADU traffic in until SBE receives a command to open the unsecure window from the host
  - Note that the SBE will use PBA bar 2
- Set HRMOR to point node address + 130MB

5.2 proc_sbe_instruct_start

a p9_sbe_instruct_start.C

- Start instructions on one core, one thread
- Thread 0 will be started at CIA scan flush value of 0x0
  - With HRMOR this is address 130MB
- Instruction start on one core, one thread. After executing this istep the SBE will load its runtime ChipOps

6 Step 6 Hostboot – Master Init, discovery

6.1 host_bootloader (non-steppable istep)

- Boot loader needs the following information:
  - LPC base address
  - Xscom base address
  - Which PNOR side it is booting from
- Perform any LPC setup (via MMIO only)
- Boot loader finds the FFS partition table in PNOR, locates the HBB partition
- Performs dcbz of HBB destination (128MB) for 512KB
- Loads HBB w/ECC to secure memory (4MB relative)
- Remove ECC to secure memory (5MB relative)
- Uses signature validation code to validate (@ 5MB relative)
- Copy down verified image to 128MB
- Copy down security algorithm, hash of the HW keys, HBB header
- Starts executing at 128MB (sets HRMOR and jumps)
- If any of the above steps fail – bootloader will checkstop the system

6.2 host_setup (non-steppable istep): Setup host environment

- If in secure boot the bootloader has already validated image
- Select primary thread (only thread running)
- Purge the L3 of all areas except for hostboot base image
- Dcbz in the Hostboot memory footprint
- Initial setup
  - stacks
  - MSR
  - execution environment
  - Thread control structures
  - Memory Management setup
- Ready for execution
  - Tracing

- Device Drivers
  - Xscom (Scom)
  - Mailbox (Scom)
  - I2C (Scom)
  - LPC
  - FSI (Scom)

♦ At this point the HWPF is alive and active

♦ p9_thread_control.C

- Start and release all other threads on core (1-3)

♦ Hostboot will pull appropriate scratch register data and write into ATTR

- Specifically the next bucket and boot flags (maybe share some code with SBE HWP?)

♦ HB mechanism to read/write to PNOR

- Host writes to LPC ↔ SPI NOR controller to read/write
- SBE uses NOR at lowest frequency, Hostboot will use flash config info to speedup to full frequency

6.3 host_istep_enable (non-steppable istep): Hostboot istep ready

♦ Hostboot checks PNOR/SIO registers (BMC) for istep attribute, if set Hostboot "halts" and waits for commands from SP

♦ Only isteps after this point can be issued to Hostboot

♦ At this point communication can be performed with the SP

6.4 host_init_bmc_pcie : Setup the PCIE to the BMC chip

♦ This chip is a no-op and is left as a placeholder if PCIe logic is desired early in the boot

♦ Required that System topology has BMC attached to master processor, otherwise this step cannot be done.

6.5 host_init_fsi : Setup the FSI links to slave chips

♦ It is expected that the following steps have already been done by SP – Hostboot will just use FSI bus

- Configure FSI master (HUB and Cascade)
- Send break commands to FSI slaves

- Configure the slaves
- Force lbus

♦ Setup Scom device drivers

- Read ID/EC levels

♦ Reset all I2C engines/slaves on the P8 Master Chip and all FSI I2C Masters (P8 slaves, centaurs)

- Can't reset the scom only I2C master on the P8 Slave chips (see 8.44)

6.6 host_set_ipl_parms : Build ipl parameters

♦ Sets the IPL parameters for this boot

6.7 host_discover_targets : Builds targeting

♦ Determines what targets are present and functional

♦ This is the step where the host "configures" itself and builds its present/functional map of the targets

- Uses FSI presence to detect processors and memory buffers
- Reads dimm VPD from PNOR/I2C to determine what dimms are present

♦ For OpenPower systems Hostboot will push the IPMI FRU inventory to the BMC

- Must push for all present parts
- Must update FRU present/functional state

6.8 host_update_master_tpm : Update the Master TPM

♦ If redundant TPM this step must enforce that master/alt-master use their local respective TPM

- If the master proc's TPM is not functional, force a reboot to the Alt Master

♦ Perform the TPM Initialization

♦ Extend TPM with measurements and configuration data

- SBE, Hash of HW public keys, HBB, HBI, etc
- See/update with list in Tim's Doc

6.9 host_gard : Do Gard

♦ Run PRD analysis of previous boot FIRDATA if present to see if something needs to be deconfigured/garded

♦ Apply repeat-gard records and deconfigure hardware

♦ Initialize PRD

- At the end of this step ATTN/PRD will start polling for errors on the master chip

6.10 host_revert_sbe_mcs_setup : Clean up MCS regs

a p9_revert_sbe_mcs_setup.C

- Clean up the MCS BARs that were used by SBE and Hostboot to cleanly load/purge the L3 cache
- Re-enable speculative reads

6.11 host_start_occ_xstop_handler : Start OpenPOWER xstop

- Image is loaded from PNOR
- Put a very small bootloader into mainstore
- FIR Master/FIR DATA is updated directly into SRAM
- OCC is started (occ_control)

6.12 host_voltage_config : Calculate correct chip voltage

- This step will compute and store all of the various system frequencies and voltages – specifically the powerbus and core frequency based on MRW wattage/powerbus frequency settings
- The programmable voltages for each P9 socket in the system (VCS, VDN, VDD) will also be calculated. The VDN and VDD rails are always on the AVS bus because the OCC needs to dynamically manipulate for Workload Optimized Frequency, but the VCS can be connected differently based on system type.

b p9_setup_evid.C (COMPUTE)

- Use VPD backed attributes (from #V) to calculate VDD, VCS and VDN for this socket
- These need to be stored to ATTR_*_VAL (VCS, VDD, VDN)

- Note that none of the settings are written to hardware – this is done later in the boot.

7 Step 7 Hostboot – MC Config

Note that the "FW Reconfig" loop starts here (since it doesn't touch HW). Any reconfig during step 7 will loop back to this step

7.1 host_mss_attr_cleanup : Spot to clean up ATTR

a p9_mss_attribute_cleanup.C (list of all mcs)

- Called on all present memory buffers (Nimbus and Centaur)
- Hook to clean up attributes on reconfig loop (set to known state) if needed

7.2 mss_volt : Calc dimm voltage

a p9c_mss_volt.C (vector of centaurs)

b p9_mss_volt.C (list of functional mcs)

- Procedure is called all the dimms on a voltage rail
- Calculate rail Voltage and updates rail system attribute
- Save settings in variables (saved in framework/cache)
- Procedure handles checking overrides

c p9c_mss_volt_avdd_offset.C (vector of centaurs)

d p9c_mss_volt_vcs_offset.C (vector of centaurs)

e p9c_mss_volt_vdd_offset.C (vector of centaurs)

f p9c_mss_volt_vddr_offset.C (vector of centaurs)

g p9c_mss_volt_vpp_offset.C (vector of centaurs)

7.3 mss_freq : Calc dimm frequency

a p9c_mss_freq.C (centaur)

- Called on each centaur

b p9_mss_freq.C (functional mcs)

- Procedure is called on each MCS in the system
- Looks at voltage and dimm functionality
- Takes a system ATTR that defines the allowable dimm frequencies for the system
- Bound frequency base on plug rules
- Calculate per memory controller frequency from attributes – picks the frequency bucket to use
- Save settings in variables (saved in framework/cache)
- Procedure handles checking overrides

c p9_mss_freq_system.C (all functional mcbists) -- Nimbus only

- Determine the optimal system nest frequency, synchronous mode is preferred
  - All dimms must be at same frequency as system
  - Otherwise move nest to max frequency defined by system and run in async mode
  - Outputs a synchronous mode ATTR and desired nest freq

- FW examines current synchronous mode and nest freq and will customize the SBE and reboot if necessary on the master only (slaves get data via mbox scratch registers)
  - ***p9_xip_customize.C***
  - Cronus may output error and stop if freqs don't match

7.4 mss_eff_config : Determine effective config

a p9c_mss_eff_config.C (mba) -- loop over all functional mba

b p9_mss_eff_config.C (mcs) -- loop over all functional mcs

- Decode SPD
  - getDimmSPD(DIMM)
  - getVPD (MCS, MR, <freq>) – need effective dimm freq for this mcs
  - getVPD (MCS, MT, <numranks for dimm0, numranks for dimm1>)
    - need number of ranks for dimms behind this mcs (effective) (dimm0=outside dimm, dimm1=inside dimm)

c mss_eff_mb_interleave.C (Cumulus only)

- Called on each centaur target.
- This sets up the MBA interleaving internal to the centaur

d p9c_mss_eff_config_thermal.C (mba) -- loop over all functional mba

e p9_mss_eff_config_thermal.C (mcs)

- getVPD(MCS, MV, ???like MT???)
- getVPD(MCS, MW, ???like MT???)
- Perform thermal calculations for the effective config

f p9_mss_eff_grouping.C (proc chip) – loop over all functional (Cumulus and Nimbus both)

- Called on each P9 target.
- Maps memory behind each chip

7.5 mss_attr_update :MSS ATTR Overrides

a p9_mss_attr_update.C

- Called per MC
- Stub HWP for FW to override attributes programmatically

8 Step 8 Hostboot – Nest Chiplets

8.1 host_slave_sbe_config

- Need to run this from master processor to all slave processors for Secureboot hole (need to ensure that SP didn't leave compromised P8 Slave.

b p9_setup_sbe_config.C

- Update SBE config data area with any configs/parameters required by SBE (see step 0 for more details)
- This includes the nest (and memory frequency if in synchronous mode)
- Configuration flags (MPIPL, etc)

8.2 host_setup_sbe

a p9_set_fsi_gp_shadow.C

- Done for all boots – some settings will change based on system type and IPL type
- Set the GP bits to default state
- Needs to take into account to not change values set up in p9_set_clock_term.C procedure

8.3 host_cbs_start

a p9_start_cbs.C

- Set a bit to start the SBE engine on master chips. Located in FSI GP region
- This same bit performs the scan0 flush of pervasive

8.4 proc_check_slave_sbe_seeprom_complete : Check Slave SBE Complete

- Check to make sure that the slave SBE engines have completed their IPL
- FW will poll for up to 1 second to see if the "done" signature is in the status reg (not tied to istep number)
- If "done" signature is not found then FW must extract FFDC from the SBE

b p9_get_sbe_msg_register.C

- Read the SBE state reg

c p9_extract_sbe_rc.C -soft_err

- Called on slave chips to look for any correctable errors on the PNOR and/or SEEPROM
- The soft_error flag just tells the procedure to not generate an error if no HW issue

d Reset all scom only I2C engines/slaves on the P8 Slave Chips

8.5 host_attnlisten_proc : Start attention poll for P9(s)

- Enable hostboot to start including all processor attentions in its post istep analysis
- Enable OCC to collect FIR data on all processors if master processor checkstops
- From this point on ATTN/PRD will listen ("poll") for powerbus attentions after each named istep

8.6 host_p9_fbc_eff_config : Determine Powerbus config

a p9_fbc_eff_config.C (None)

- Sets system wide attributes derived from MRW and system topology

- Epsilon settings
- Processor floor frequency

♦ Does not access the HW

8.7 host_p9_eff_config_links : Powerbus link config

a p9_fbc_eff_config_links.C (None)

♦ Determines the Sets system wide attributes derived from MRW and system topology

- Epsilon settings
- Processor floor frequency

8.8 proc_attr_update :Proc ATTR Update

a p9_attr_update.C

♦ Called per processor

♦ Stub HWP for FW to override attributes programmatically

8.9 proc_chiplet_scominit : Scom inits to all chiplets (sans Quad)

a p9_chiplet_fabric_scominit.C

♦ Initfiles in procedure defined on VBU ENGD wiki

♦ Apply scom overrides to all chiplets necessary to init the powerbus

- p9.fbc.no_hp.scom.initfile
- p9.fbc.ioe_dl.scom.initfile
- p9.fbc.ioe_tl.scom.initfile
- p9.fbc.ioo_dl.scom.initfile
- p9.fbc.ioo_tl.scom.initfile

8.10 proc_xbus_scominit : Apply scom inits to Xbus

a p9_xbus_scominit.C

♦ Each instance of bus must have unique id set for it – personalize it

♦ Must set present and valid bits based on topology (Attributes indicate present and valid)

8.11 proc_chiplet_enable_ridi : Enable RI/DI for xbus

a p9_xbus_enable_ridi.C

♦ Drop RI/DI for xbus chiplets being used

♦ Any other chip wide RI/DI

9 Step 9 Hostboot – EDI+ and Electrical O-Bus Initialization

9.1 fabric_erepair : Restore Fabric Bus eRepair data

a *p9_io_restore_erepair.C (O, X bus target pairs)*

- Restore/preset bad lanes on electrical O and X buses from VPD (in drawer)
- Applies powerbus repair data from module vpd (#ER keyword in VRML VWML)
- Runtime detected fails that were written to VPD are restored here
- NOOP for Cronus

9.2 fabric_io_dccal : Calibrate Fabric interfaces

a *io_dccal.C (O, X bus target pairs passed in)*

- Will be called per bus target pair
- Calibration of TX impedance, RX offset for O and X busses
- Needs to be quiet on the bus – drivers are quiesced and driving 0s – O, X buses
- Must be complete on ALL chips before starting O, X bus training
- Expect to use a calculation (floating point)
- At end of offset calibration there may be a lane that is bad
  - FW must record bad lane and write to VPD for future eRepair (handled when PRD starts)
  - Must generate error log, procedure will mark lane bad in HW (which future procedure take advantage of)

9.3 fabric_pre_trainadv : Advanced pre training

a *p9_io_pre_trainadv.C (called on each O and X bus target pair)*

- Debug routine for IO Characterization
- Nothing in it

9.4 fabric_io_run_training : Run training on internal buses

a *p9_io_xbus_linktrain.C (called on each OO and X bus target pair)*

- Hostboot will run training on all intra node buses. For Nimbus this is all X buses. For Cumulus this is run by the SP in a later step
- Wiretest, Deskew, Eye Optimization, and repair
  - Option to run extend bit patterns in optimization phase (replaces RDT)
  - Repairable fails are left for PRD to analyze and move data into VPD

- PRD will use ***io_eRepair_read.C*** to perform this

- Fatal bus training errors are handled by procedure, must return error and FFDC (written to VPD)

- Expected that fatal error passes returncode back to HWPF, FW then looks up returncode and determines what to do based off of FFDC

9.5 fabric_post_trainadv : Advanced post EI/EDI training

a p9_io_post_trainadv.C (called on each O and X bus target pair)

- Debug routine for IO Characterization

- Nothing in it

9.6 proc_smp_link_layer : Start SMP link layer

a p9_smp_link_layer.C (called on processor chip)

- Reads logical A/X link configuration attributes, trains the DL/TL layers of selected links

- Set scom on both sides of the bus to trigger Data link layer training

- DLL sends training packets, sets link up FIR bit when done

- FIR done bit launches the Transaction Layer (TL)

- FIR bit in nest domain to indicate training done

- After this point the mailbox register are available to communicate

  - Xstop would prevent mailbox communication

- Bus is NOT part of the SMP coherency

- Only performed on trained, valid buses

9.7 proc_fab_iovalid : Lower functional fences on local SMP

a p9_fab_iovalid.C (chip target)

- Reads logical A/X link config, sets iovalid for selected links

- Only performed on trained, valid buses

- After this point a checkstop on a slave will checkstop master

- Reads the A/X link delays for later HWP to pick best link for coherent traffic

9.8 host_fbc_eff_config_aggregate : Pick link(s) for coherency

a p9_fbc_eff_config_aggregate.C (chip target)

- Reads attributes from previous HWP and determines per-link address/data capabilities

- Sets up attributes for build SMP

10 Step 10 Hostboot – Activate PowerBus

10.1 proc_build_smp : Integrate P9 Islands into SMP

a *p9_build_smp.C (vector of all chips to include in SMP)*

- Look for checkstops
- Use the fabric concurrent maintenance operation to merge P9 PB islands into the SMP
- Fabric config between IO/CAPI are set here – only can set once, must be known by this point in time
- After this point the SMP is built for normal mode
- Runs initfiles to set current/next values for full config in slaves, setup master next value
  - p9.fbc.ab_hp.scom.initfile
  - p9.fbc.cd_hp.scom.initfile
- Trigger fabric quiesce/switch/init on the master

10.2 host_slave_sbe_update

- On systems that support Alt Master Processors then code will attempt to read the TOC of the Alt Master PNOR to check for connection problems. If an error is detected it will be logged, but this does not stop the IPL (except when in manufacturing mode)
- Hostboot must update SEEPROM because the SP cannot because of secureboot. It is at this step in the IPL so it can be updated via Xscom (trusted path) on all chips in the system

b *p9_customize_image.C*

- If needed build a custom SEEPROM image for each chip in the system off of the base IPL SEEPROM image
- This set will update all SEEPROM images in the HB "node". All needed attributes are written from the host into the SBE image via this HWP.
- In addition if the override section from the PNOR is not empty then it needs to be appended to the SBE image prior to customization.

- If the SEEPROM was updated then Hostboot will request a reipl at this point

10.3 host_set_voltages : Set correct chip voltage(s)

- This step will apply the voltages calculated earlier in the IPL. It is done here so all chips can be programmed at one spot.

b *p9_setup_evid.C (APPLY_AVS)*

- Via the AVS bus the HWP will program always program VDN and VDD. The specific combination of AVS bus and rail select are indicated by ATTR_*_BUS_CTL (which AVS bus) and ATTR_*_BUS_SELECT (which select).

- The VCS voltage will be programed if ATTR_VCS_BUS_CTL indicates an AVS bus (taking into account the rail select as well), but if set to I2C or SP then it is not programmed.

♦ If ATTR_VCS_BUS_CTL indicates that it is programed via non AVS bus means then Hostboot then needs to use the value in ATTR_VCS_VAL to program (via direct I2C or message to FSP/BMC). The Attributes needed for these Hostboot operations (I2C bus information, FSP control) are assumed to be part of the MRW

10.4 proc_cen_ref_clk_enable : Setup centaur ref clocks

a p9_cen_ref_clk_enable.C (Cumulus only)

♦ Enable the ref clocks to centaur

10.5 proc_enable_osclite

a p9_enable_osclite.C

♦ Cumulus only

♦ Turn off the power-pon-reset to osclite macro

♦ Setup oscillator mode based on istep 0 setup

♦ Check that osclite matches expected output (if not returns an error for FW to trigger reconfig)

10.6 proc_chiplet_scominit : Scom inits to all chiplets (sans Quad)

a p9_chiplet_scominit.C

♦ Initfiles in procedure defined on VBU ENGD wiki

♦ Apply scom overrides to all good chiplets (except EX and MC)

- p9.fbc.no_hp.scom.initfile

b p9_psi_scominit.C

♦ Each instance of bus must have unique id set for it – personalize it

♦ Must set present and valid bits based on topology (Attributes indicate present and valid)

10.7 proc_abus_scominit : Apply scom inits to Abus

a p9_abus_scominit.C

♦ Each instance of bus must have unique id set for it – personalize it

♦ Must set present and valid bits based on topology (Attributes indicate present and valid)

10.8 proc_obus_scominit : Apply scom inits to Obus

a p9_obus_scominit.C

♦ Each instance of bus must have unique id set for it – personalize it

♦ This is where the O to A/NVlink linkage is setup in HW

10.9 proc_npu_scominit : Apply scom inits to NPU

a. ***p9_npu_scominit.C***

- Each instance of NPU bus must have unique id set for it – personalize it

10.10 `proc_pcie_scominit : Apply scom inits to PCIe chiplets`

a. ***p9_pcie_scominit.C***

- Initfiles in procedure defined on VBU ENGD wiki
- Perform the PCIe Phase 1 Inits 1-8
  - Sets the lane config based on MRW attributes
  - Sets the swap bits based on MRW attributes
  - Sets valid PHBs, remove from reset
  - Performs any needed overrides (should flush correctly) – this is where initfile may be used
  - Set the IOP program complete bit
  - This is where the dSMP versus PCIE is selected in the PHY Link Layer

10.11 `proc_scomoverride_chiplets : Apply sequenced scom inits`

a. ***p9_scomoverride_chiplets.C***

- Apply any sequence driven scom overrides to chiplets – Should be NONE

10.12 `proc_chiplet_enable_ridi : Enable RI/DI chip wide`

a. ***p9_chiplet_enable_ridi.C***

- Drop RI/DI for all chiplets being used (A, O, PCIe, DMI)
- Any other chip wide RI/DI

10.13 `host_rng_bist : Trigger Built In Self Test for RNG`

a. ***p9_rng_init_phase1.C***

- Trigger the Random Number Generator Built In Self Test (BIST). Results are checked later in step 16 when RNG is secured

10.14 `host_update_redundant_tpm : Update the Alt Master TPM`

- Perform hostimprint of both master/alt-master TPM/SEEPROM if needed
  - If non-functional TPM during hostimprint then fail IPL
    - Must clear both redundant TPM before updating SEEPROMs
  - When hash of HW public keys is updated in SEEPROM, must clear the software root key in TPMs
  - If imprint is done then reboot

- Replay information from master TPM into alternate TPM

b p9_update_security_ctrl.C

- This HWP will set the SUL security bit so that SBE image cannot be updated
- This will also make the SAB security bit read only
- If a TPM is non functional, set the TDP (TPM Deconfig Protection) to prevent attack vector

11 Step 11 Hostboot Centaur Init

The following steps are part of the Centaur initialization. Unlike P8, individual sub-steps can be done in both Hostboot and Cronus.

Hostboot will check for HW reconfig loop after the end of each named istep. For P9 Scale Out if a reconfig loop is detected then Hostboot will request a reboot from the service processor during any step except for istep 7. For P9 Scale Up if a reconfig request happens in 11,12,13, or 14 it will go back to the beginning of step 7, then redo steps 11,12,13 (known as HW reconfig loop). If it fails in step 7 it will go back to the beginning of step 7 (FW reconfig loop).

Note that this is the step for HW Reconfig to restart on Centaur or dimm training/init fails

```
11.1 host_prd_hwreconfig        : Hook to handle HW reconfig
```

- This step is always called
- Move all Centaur's inband scom back to FSI scom
- Call PRD to allow them to rebuild model to remove non-functional Centaurs
- Protect Centaur from SP operations during initialization
  - Set the CFP Security bit. This will prevent the SP from performing FSI operations to the Centaur while it is being initialized
- Used for HW reconfig path. FW's strategy is to perform the reconfig on ALL functional Centaurs/MCS's in the system.
- The following procedures must be called:

b p9_switch_cfsim.C (proc target)

- Call on all present processors
- Move all Centaur’s inband scom back to FSI scom

c p9_enable_reconfig.C (MCS, DMI, MCA/MBuf)

d Call on all present MCS targets

- Enables HW for reconfig loop
- Cumulus/Centaur:
  - Attribute (ATTR_CEN_MSS_INIT_STATE) to each Centaur to track where the Reconfig loop got to:
    - Clocks on (can do fir masking) – set after step 11
    - DMI bus up (inject special bit) – set after framelock
      - Turn's on special bit that allows the MCS DMI to get errors and not get into a hang condition
      - Mask a bunch of FIRs on processor
      - Mask a bunch FIRs on centaur (HWP will check clock state)
      - Injects a fail on the DMI bus (only if DMI bus is alive)
      - Clears IO/MCS FIRs
      - Turns off special bit
- Nimbus
  - Raise the MCU chiplet fences
  - Stop clocks
  - Scan 0 flush the MCU chiplet each and everytime through this loop
  - How do we cleanup the nest portion of the MCS?

The following steps are for Cumulus only. They are not defined/applicable for P9 Nimbus

11.2 cen_tp_chiplet_init1 : Centaur TP chiplet init, stage #1

a cen_tp_chiplet_init1.C (MemBuf)

- Flush all GP registers content to default state
- Drop fences, check VDD, start VITL clocks
- Scan 0 PLL GPTR/BNDY/FUNC rings.

11.3 cen_pll_initf : Program Nest PLL

a cen_pll_initf.C

- Apply the TP BNDY PLL ring with setpulse. This includes settings for NEST/MEM/DMI (cleanup) PLLs
  - Final frequency is known at this point – DDR is @ 1600
  - Nest freq: 2400MHz

11.4 cen_pll_setup : Setup Nest PLL

a cen_pll_setup.C (MemBuf)

- Performs PLL checking
- The memory PLL (ie DDR4) are set to the correct speeds for both DDR3 and DDR4 (1600)
- Establish Nest PLLs (feeds TP chiplet) and MEM PLL

11.5 cen_tp_chiplet_init2 : Centaur TP chiplet init, stage #2

a cen_tp_chiplet_init2.C (MemBuf)

- Scan 0 init TP unit flush
- Start PIB/NET clock
- Invoke Repair Loader
- Writing FSI GP3 to switch mux
- No repair/timing for TP chiplet (i.e. fuses).

11.6 cen_tp_arrayinit : Centaur TP chiplet array init

a cen_tp_arrayinit.C (MemBuf)

- Run arrayinit on TP chiplet, when done, all arrays are initialized
- Scan flush 0 to all rings except GPTR, Time, and Repair

11.7 cen_tp_chiplet_init3 : Centaur TP chiplet init, stage #3

a cen_tp_chiplet_init3.C (MemBuf)

- Start clock on PERV region
- Enable PIB trace mode,
- When done, the TP chiplet can be used to init the rest of the chip. All access now go through TP chiplet

The following steps are to initialize the Centaur chip logic (Host Chiplet Setup)

11.8 cen_chiplet_init : Centaur chiplet init

a cen_chiplet_init.C (MemBuf)

- Identify good chiplets then for each good chiplet:
  - Setup multicast groups
  - Scan 0 all rings
  - If repair ring is present, kick off the fuse repair algorithm (load repair ring)
  - DTS calibration via repair loader. Repairs are loaded from OTPROM fuse.
    - Pulls data from OTPROM and puts into repair ring (series of Scoms)
    - No actual ring content from VPD

11.9 cen_arrayinit : Centaur chiplet array init

a cen_arrayinit.C (MemBuf)

- Run arrayinit on all good chiplets, except for TP chiplet. After this, all chiplet arrays are initialized
- Scan flush 0 to all rings, except GPTR, Time, and Repair.

Note:

- If LBIST was to be run, it should be run after this step, prior to the next step

The following sections are to initialize the Centaur chip logic (Host Chiplet Initialization)

11.10 cen_initf : Centaur Scan overrides

a ***cen_initf.C (MemBuf)***

- Perform any scan overrides for Centaur
  - May not have any config dependent scans
- Does not include the pervasive region

11.11 cen_do_manual_inits : Manual Centaur Scans

a cen_do_manual_inits.C (MemBuf)

- Currently empty (Thermal Init has been moved to cen_initf.C. Disabling cache has been moved to repair loader)
- Perform any non-initfile scan overrides for Centaur
- Should be avoid, place holder for workaround only.

The following sections are to initialize the Centaur chip Scom logic

11.12 cen_startclocks : Start Centaur Nest/MEM clocks

a ***cen_startclocks.C (MemBuf)***

- Starts Centaurs' NEST and MEM chiplet clocks. This includes the L4, DMI, DDR, and MBA clocks.
- Deassert the memrst_b GP bit to activate the reset_OE signal
- Enable driver and receivers (set appropriate GP bits)
- Lower RI and DI inhibits

11.13 cen_scominits : Perform Centaur SCOM inits

a ***cen_scominits.C (MemBuf)***

- Currently empty.
- Any needed scom initializations - no config dependent settings allowed

12 Step 12 Hostboot – DMI Training

The following steps are for Cumulus only. They are not defined/applicable for P9 Nimbus

12.1 mss_getecid : Read out ECID of all Centaurs

a ***p9c_mss_cen_getecid.C (Centaur)***

- Sets ATTR_CEN_MSS_INIT_STATE to "clocks on"
- Read the ECID for each centaur and store away for callouts.
- Decode ECID and set other ECID related attributes for later operations on Centaurs

12.2 dmi_attr_update : Update DMI related attributes

a ***p9_io_dmi_attr_update.C (void)***

- Currently empty.
- Attribute targets: MCS/MemBuf
- Stub HWP for FW to override attributes programmatically.

12.3 proc_dmi_scom_init : DMI Scom setup on Cumulus DMI

a p9_io_dmi_scominit.C (DMI)

- Perform scom inits for DMIs on the processor.

12.4 cen_dmi_scominit : DMI Scom setup on Centaur

a p9_io_cen_scominit.C (MemBuf)

- Perform scom inits for DMI on Centaur.

12.5 dmi_erepair : Restore EDI Bus eRepair data

- Bad lanes are preset on the receive side.

b p9_io_dmi_restore_erepair.C (DMI, vector of RX bad lanes, vector of TX bad lanes)

- Procedure that perform repairs on DMI bus (P9 side)

c p9_io_cen_restore_erepair.C (centaur, vector of RX bad lanes, vector of TX bad lanes)

- Applies centaur data from planar prom (planar centaurs), centaur dimm
- Runtime detected fails that were written to VPD are restored here

12.6 dmi_io_dccal : Calibrate DMI interfaces

a p9_io_dmi_dccal.C (DMI target)

- Calibration of TX impedance, RX offset for memory buses
  - Needed for EDI buses on p9
- Needs to be quiet on the bus – drivers are quiesced and driving 0s – EDI buses
- Must be complete on ALL Centaurs for this PgP island before starting next EDI training (host based sync point)
- At end of offset calibration there may be a lane that is bad
  - FW must record bad lane and write to VPD for future eRepair (handled when PRD starts)

b p9_io_cen_dccal.C (Centaur target)

- Calibration of TX impedance, RX offset for memory buses
  - Needed for EDI buses on Centaur
- Needs to be quiet on the bus – drivers are quiesced and driving 0s – EDI buses
- Must be complete on ALL Centaurs for this PgP island before starting next EDI training (host based sync point)
- At end of offset calibration there may be a lane that is bad
  - FW must record bad lane and write to VPD for future eRepair (handled when PRD starts)

12.7 dmi_pre_trainadv : Advanced pre-DMI training

a p9_io_dmi_pre_trainadv.C (DMI/ Centaur pair)

- Currently empty
- Debug routine for IO Characterization

12.8 dmi_io_run_training : Run training on MC buses

a p9_io_dmi_linktrain.C (DMI/Centaur pair)

- Train internal DMI bus
- Wiretest, Deskew, Eye Optimization, and repair
  - Option to run extend bit patterns in Optimization phase (replace RDT)
  - Wiretest fails are left for PRD to analyze and store data into VPD
  - Fatal bus training errors are handled by HWP and written to VPD

12.9 dmi_post_trainadv

a p9_io_dmi_post_trainadv.C (DMI/Centaur pair)

- Currently empty
- Debug routine for IO Characterization

12.10 proc_cen_framelock : Initialize EDI Frame

a ***p9_cen_framelock.C (DMI/Centuar pair)***

- Raise IO Valid – Allow link init traffic (scrambled patterns) on EDI bus
- P9 Centaur initial frame lock
  - Starts listening automatically after IOValid raised
  - Started on the P9 logic
  - If a bit error (CRC) in the middle need to re-FrameLock
- Round trip delay calculation
  - Host code can trigger and check
- When done, Inband accesses are now viable
- Hardware xmitting idle frames
- Enabled CRC checking
- EDI is at runtime state
- If successful, set ATTR_MSS_INIT_STATE to DMI active on Centaur

12.11 host_startprd_dmi : Load PRD for DMI domain

- Currently empty

12.12 host_attnlisten_memb : Start attention poll for membuf

- Currently empty
- Expand Host PRD to include memory buffers (as well as powerbus)
- Enable OCC to collect FIR data on all memory buffers if master processor checkstops

12.13 cen_set_inband_addr : Set the Inband base addresses

a ***p9c_set_inband_addr.C (proc Chip Target)***

- Any initializations to setup Inband access path.
  - MI – Scom base address for each contained DMI bus
  - Centaur – any other settings

- ALL ACCESES from this point on in are Inband access for Centaur unless otherwise specified

13 Step 13 Hostboot – DRAM Training

13.1 host_disable_memvolt : Disable VDDR on Warm Reboots

a *Power off dram – VDDR and vPP. Must drop VDDR first, then VPP.*

- Turned off here to handle reconfig loop for dimm failure
- Only really issued if VDDR/VPP is on

13.2 mem_pll_reset : Reset PLL for MCAs in async

a *p9_mem_pll_reset.C (proc chip)*

- This step is a no-op on cumulus as the centaur is already has its PLLs setup in step 11
- This step is a no-op if memory is running in synchronous mode since the MCAs are using the nest PLL, HWP detect and exits
- If in async mode then this HWP will put the PLL into bypass, reset mode
- Disable listen_to_sync for MEM chiplet, whenever MEM is not in sync to NEST

13.3 mem_pll_initf : PLL Initfile for MBAs

a *p9_mem_pll_initf.C (proc chip)*

- This step is a no-op on cumulus
- This step is a no-op if memory is running in synchronous mode since the MCAs are using the nest PLL, HWP detect and exits
- MCA PLL setup –
  - Note that Hostboot doesn't support twiddling bits, Looks up which “bucket” (ring) to use from attributes set during mss_freq
  - Then request the SBE to scan ringId with setPulse
    - SBE needs to support 5 RS4 images
    - Data is stored as a ring image in the SBE that is frequency specific
    - 5 different frequencies (1866, 2133, 2400, 2667, EXP)

13.4 mem_pll_setup : Setup PLL for MBAs

a *p9_mem_pll_setup.C (proc chip)*

- This step is a no-op on cumulus
- This step is a no-op if memory is running in synchronous mode since the MCAs are using the nest PLL, HWP detect and exits

- MCA PLL setup
  - Moved PLL out of bypass(just DDR)
- Performs PLL checking

13.5 proc_mcs_skewadjust : Update clock mesh deskew

a ***This step is a no-op***

13.6 mem_startclocks : Start clocks on MBA/MCAs

a ***p9_mem_startclocks.C (proc chip)***

- This step is a no-op on cumulus
- This step is a no-op if memory is running in synchronous mode since the MCAs are using the nest PLL, HWP detect and exits
- Drop fences and tholds on MBA/MCAs to start the functional clocks

13.7 host_enable_memvolt : Enable the VDDR3 Voltage Rail

a ***Bring power to dram rails VDDR and VPP. VPP must be enabled prior to VDDR***

- BMC based systems – this is a no-op
- Send message to FSP to turn on voltages
  - Message must have accounted for voltage/current tweaking based on number of plugged dimms (Dynamic VID)
  - Pulled from HWPF attributes per voltage rail
  - FSP
    - Trigger voltage ramp to DPSS via I2C
    - Wait for min 200 ms ramp, must be stable 500us after DPSS claims Pgood
- Wait for ack message from FSP – confirms that voltage is on and ready

13.8 mss_scominit : Perform scom inits to MC and PHY

a ***p9_mss_scominit.C (mcbist) -- Nimbus***

b ***p9c_mss_scominit.C (membuf) -- Cumulus***

- HW units included are MCBIST, MCA/PHY (Nimbus) or membuf, L4, MBAs (Cumulus)
- Does not use initfiles, coded into HWP
- Uses attributes from previous step
- Pushes memory extent configuration into the MBA/MCAs

- Addresses are pulled from attributes, set previously by mss_eff_config
- MBA/MCAs always start at address 0, address map controlled by proc_setup_bars below

13.9 mss_ddr_phy_reset : Soft reset of DDR PHY macros

a p9_mss_ddr_phy_reset.C (mcbist) -- Nimbus

b p9c_mss_ddr_phy_reset.C (mba) -- Cumulus

- ♦ Lock DDR DLLs
  - Already configured DDR DLL in scaninit
- ♦ Sends Soft DDR Phy reset
- ♦ Kick off internal ZQ Cal
- ♦ Perform any config that wasn't scanned in (TBD)
  - Nothing known here

13.10 mss_draminit : Dram initialize

a p9_mss_draminit.C (mcbist) -- Nimbus

b p9c_mss_draminit.C (mba)-- Cumulus

- ♦ RCD parity errors are checked before logging other errors – HWP will exit with RC
- ♦ De-assert dram reset
- ♦ De-assert bit (Scom) that forces mem clock low – dram clocks start
- ♦ Raise CKE
- ♦ Load RCD Control Words
- ♦ Load MRS – for each dimm pair/ports/rank
  - ODT Values
  - MR0-MR6

c Check for attentions (even if HWP has error)

- ♦ FW
  - Call PRD
    - If finds and error, commit HWP RC as informational
    - Else commit HWP RC as normal
  - Trigger reconfig loop is anything was deconfigured

13.11 mss_draminit_training : Dram training

a ***p9_mss_draminit_training.C (mcbist)-- Nimbus***

b ***p9c_mss_draminit_training.C (mba) -- Cumulus***

- Prior to running this procedure will apply known DQ bad bits to prevent them from participating in training. This information is extracted from the bad DQ attribute and applied to Hardware
  - Marks the calibration fail array
- External ZQ Calibration
- Execute initial dram calibration (7 step – handled by HW)
- This procedure will update the bad DQ attribute for each dimm based on its findings

13.12 mss_draminit_trainadv : Advanced dram training

a ***p9_mss_draminit_training_advanced.C (mcbist target) -- Nimbus***

b ***p9c_mss_draminit_training_advanced.C (mba target) -- Cumulus***

- Prior to running this procedure will apply known DQ bad bits to prevent them from participating in training. This information is extracted from the bad DQ attribute and applied to Hardware
  - Marks the MCBist mask
- This step will contain any algorithms to improve data eye post training
  - At the moment this is a no-op for P9 Nimbus
  - For P9 Cumulus the VREF calibration will be done here
- Also will contain some characterization (mfg only) tests
  - There will be a FAPI interface for dumping characterization data, platform implementation is TBD (dump to console, memory, PNOR)
- This procedure will update the bad DQ attribute for each dimm based on its findings

13.13 mss_draminit_mc : Hand off control to MC

a ***p9_mss_draminit_mc.C (mcbist) -- Nimbus***

b ***p9c_mss_draminit_mc.C (membuf) -- Cumulus***

- P9 Cumulus -- Set IML complete bit in centaur
- Start main refresh engine
- Refresh, periodic calibration, power controls
- Turn on ECC checking on memory accesses
- Note at this point memory FIRs can be monitored by PRD

14 Step 14 Hostboot – DRAM Initialization

14.1 mss_memdiag : Mainstore Pattern Testing

- The following step documents the generalities of this step
  - In FW PRD will control mem diags via interrupts. It doesn't use mss_memdiags.C directly but the HWP subroutines
  - In cronus it will execute mss_memdiags.C directly

b p9_mss_memdiags.C (mcbist)--Nimbus

c p9_mss_memdiags.C (mba) -- Cumulus

- Prior to running this procedure will apply known DQ bad bits to prevent them from participating in training. This information is extracted from the bad DQ attribute and applied to Hardware
- Nimbus uses the mcbist engine
  - Still supports superfast read/init/scrub
- Cumulus/Centaur uses the scrub engine
- Modes:
  - Minimal: Write-only with 0's
  - Standard: Write of 0's followed by a Read
  - Medium: Write-followed by Read, 4 patterns, last of 0's
  - Max: Write-followed by Read, 9 patterns, last of 0's
- Run on the host
- This procedure will update the bad DQ attribute for each dimm based on its findings
- At the end of this procedure sets FIR masks correctly for runtime analysis
- All subsequent repairs are considered runtime issues

14.2 mss_thermal_init : Initialize the thermal sensor

a mss_thermal_init.C – Cumulus/Centaur only

- Called on Centaur target,
- NOTE: On Nimbus OCC has to directly read the thermals via the I2C Masters (shared with Host code)
  - Use lock HW and FW algorithm between OCC, Hostboot/OPAL/PHYP
- Setup and configure I2C thermal sensor on dimms
- Configure and start centaur thermal cache

- Configure and start the OCC cache
- Disable safe mode throttles
  - Will cause memory to go to runtime emergency throttles
  - When OCC starts polling OCC cache will revert to runtime settings

b p9_throttle_sync.C

- Must be issued on all P9s, can only be issued after ALL centaurs on given p9 have thermal init complete (can also loop at the end of all centaurs)
- Same HWP interface for both Nimbus and Cumulus, input target is TARGET_TYPE_PROC_CHIP; HWP is to figure out if target is a Nimbus (MCS) or Cumulus (MI) internally.
- Triggers sync command from MCS to actually load the throttle values into the MBA/MCA

14.3 proc_pcie_config : Configure the PHBs

a p9_pcie_config.C

- Called on all chips, target is per PHB
- Procedural based – will call initfile if need be
- Covers PCIe Phase 2 Inits 18-30
  - Setup config regs
  - Command and Data credits
  - Clear FIRs (if needed)
  - Unmask PCIe FIRs

14.4 mss_power_cleanup : Clean up any MCS/Centaurs

a p9_mss_power_cleanup.C (mcbist) --Nimbus

b p9c_mss_power_cleanup.C (centaur, mbas) -- Cumulus

- NO-OP for Nimbus
- Called on all present Centaurs and MBAs for Cumulus
- Called on all present MCBIST for Nimbus
- Cleans up and powers down unused cenaturs/mcs/DMI
  - Hostboot will start to flow out to memory in the next step
  - Any memory errors after this point are considered “runtime errors”

- All errors from this point on have to be a no deconfig and gard OR terminate the IPL (and let the SP do the reconfig)
- If user attempts to do a deconfig outside the loop – then attempt to fail

14.5 proc_setup_bars : Setup Memory BARs

a p9_mss_setup_bars.C (proc chip) -- Nimbus

b p9c_mss_setup_bars.C (proc chip) -- Cumulus

- Same HWP interface for both Nimbus and Cumulus, input target is TARGET_TYPE_PROC_CHIP; HWP is to figure out if target is a Nimbus (MCS) or Cumulus (MI) internally.
- Prior to setting the memory bars on each processor chip, this procedure needs to set the centaur security protection bit –
  - TCM_CHIP_PROTECTION_EN_DC is SCOM Addr 0x03030000
  - TCN_CHIP_PROTECTION_EN_DC is SCOM Addr 0x02030000
  - Both must be set to protect Nest and Mem domains
- Based on system memory map
  - Each MCS has its mirroring and non mirrored BARs
  - Set the correct checkerboard configs. Note that chip flushes to checkerboard
  - need to disable memory bar on slave otherwise base flush values will ack all memory accesses

c p9_setup_bars.C

- Sets up Powerbus/MCD, L3 BARs on running core
  - Other cores are setup via winkle images
- Setup dSMP and PCIe Bars
  - Setup PCIe outbound BARS (doing stores/loads from host core)
    - Addresses that PCIE responds to on powerbus (PCI init 1-7)
  - Informing PCIe of the memory map (inbound)
    - PCI Init 8-15
- Set up Powerbus Epsilon settings
  - Code is still running out of L3 cache
  - Use this procedure to setup runtime epsilon values
  - Must be done before memory is viable

14.6 proc_htm_setup : Setup HTM allocations

a ***p9_htm_setup. C***

- Setup any BARs and inits to enable hardware in memory trace
- TBD – where does CHTM go? DD2.0 feature.

14.7 proc_exit_cache_contained : Execution from memory

a ***p9_exit_cache_contained. C***

- Allow execution to flow out to memory
- Data rolls out to memory

14.8 host_mpipl_service : Perform MPIPL tasks

- This is a no-op for warm/cold IPLs. See description in REF LOC for full details

15 Step 15 Hostboot – Build STOP Images

15.1 proc_set_pba_homer_bar :Set HOMER location in OCC

a ***p9_pm_set_homer_bar.C(uint64_t p_homer_region, …)***

- Called for each processor chip.
- Parameter: Physical address within HOMER image where OCC code will be loaded; STOPGPE image is this value + 1MB (not a pointer address, it cannot be dereferenced)
  - **NOTE**: HOMER is a 4MB region that is allocated to start 1MB ***before*** the value passed to this procedure!! This done to allow the OCC boot from the 0 offset of the PBA BAR0 value (which has a granularity of 1MB while the Core Self-Store portion must be aligned to a 2MB boundary. Additionally, the OCC complex has no need to address the first 1MB of HOMER --- only the last 3MB.
- Parameters: PBA BAR number, OCC complex HOMER image size(3MB), STOPGPE image location (default: mem; others: L3)
- ***p9_pm_pba_bar_config.C*** (called as subroutine)
  - Set BAR address

15.2 host_build_stop_image : Build runtime STOP images

- Pull Reference Image from PNOR
  - Run through secure boot algorithm

b ***p9_hcode_image_build.C (void* reference_image, void* v_homer_region, enum image_bld) FAPI2::ReturnCode***

- HOMER – Hardware Offload Microcode Engine Region
- Called for each processor chip.

- Parameter: Pointer to Reference image.
- Parameter: Pointer to Output HOMER location (virtual address). The procedure places the respective images (eg SGPE, CME) into HOMER at the appropriate offsets
  - This is any Hostboot specified mainstore location (does not have to be attached to the processor being STOPped).
  - When PHYP is loaded, the HOMER region will be trampled, PHYP will call ***p9_hcode_image_build.C*** to recreate them in a PHYP specified location in mainstore (each image will probably be placed in mainstore local to its associated processor for performance).
  - OPAL keeps same location, requires that it is at the top of memory
- Parameter: image_bld – which images to update – either PSTATE, STOP, or both
- Fused vs Normal
  - System ATTR defines, TBD on mechanism
  - Greg to work out details, likely two different rings in reference image or some RS4 merge capability
- Customize image with data for each core
  - Scan rings – Time, GPTR, Repair
  - Tweak to make runtime acceptable – expect to be only scom registers
- Write image to the appropriate offset based on the output pointer parameter

c Cronus will load the images via putmemproc

d p9_stop_gen_cpu_reg(void* v_homer_region, …)

- API that updates a STOP image with various core state registers (MSR, HRMOR, LPCR)
  - The core registers are set to these values on STOP 15 exit
- This will only be called by Hostboot. Cronus will not use it. Hence separate from ***p9_hcode_image_build.C .***

15.3 host_start_stop_engine : Initialize the STOPGPE engine

a p9_pm_stopgpe_init(chip_target, ENUM:PM_INIT) □ FAPI2::ReturnCodeCalled for each processor chip

- Parameters: PM_INIT (to perform initialization vs PM_RESET that is used during the OCC reset flow)
- Starts the Stop GPE engine
  - Bootloader runs from HOMER OCC offset + 1MB (2MB from HOMER base)
    - Copies STOP image from HOMER to OCC SRAM

- Restarts from OCC SRAM
- PK initialization -> STOP Thread(s) started
- Sets flag in OCC Flag reg that initialization is complete for HWP to poll on

♦ Loop over all functional cache chiplets

- ***p9_pfet_init.C (cache target, PM_INIT) (called as a subroutine)***
  - Initialize PFET controller parameters (delays
  - Note: this the default of the PFETs is OFF and this action will have them remain off.

♦ Loop over all functional core chiplets

- ***p9_pfet_init.C (core target, PM_INIT) (called as a subroutine)***
  - Initialize PFET controller parameters (delays)
  - Note: this the default of the PFETs is OFF and this action will have them remain off.
- **NOTE:** CME initialization is performed upon STOP exit of the cache chiplet by the STOPGPE so as to allow the wake up of any core within a Quad. This is NOT done via HWPs.

15.4 host_establish_ex_chiplet : Select Hostboot core

a p9_update_ec_eq_state.C ()

♦ Need to update multicast groups for all cores beyond the master core

- need to add each EC multicast group 0, 1
- need to add each EQ to multicast group 0

♦ Use the functional state to find all good cores

♦ Write all EQ/Core good mask into OCC complex

♦ This is the “master record“ of the enabled cores/quad in the system for runtime

16 Step 16 Hostboot – Core Activate

16.1 host_activate_master : Activate master core

♦ Hostboot sends a message to the SBE to enter the deadman loop for exit STOP15 (passes a parameter to indicate the wait time)

- Hostboot will block and wait for PSU SBE interface return

- Hostboot command will trigger the SBE to run the following HWP in its Chip OP thread (this will block SP chipOp until it either passes or triggers the checkstop)
    - SBE Deadman timer starts upon receiving the ChipOp (SBE FW handling of deadman message)
    - SBE starts timer based on ChipOp parameters
    - SBE FW will repeatedly call the following HWP to check for STOP 15 state
    - p9_sbe_check_master_stop15.C (passed in time(from PIBMEM or via Cronus)
    - Monitor master STOP 15. It can return three different values:
    - Checks for STOP 15 entered (completely entered)
    - STOP 15 reached (success) – FAPI2 SUCCESS
    - STOP 15 not reached, but no error HW state (still in progress) -- STOP15_PENDING
    - STOP 15 not reached, but HW error (failure) – any other FAPI2 RC
    - The RC and FFDC from this HWP needs to be saved by the SBE into async ChipOp FFDC space
    - SBE will set an "async FFDC" bit in the SBE status register. When the SP recognizes that the master STOP cycle failed, it can then request the "async FFDC"
    - On success SBE FW will trigger STOP 15 exit on thread 0 on the master core using . the PSU Interrupt (Separate bit in PSU doorbell)
        - In addition ***p9_block_wakeup_intr.C –clear*** must also be called to allow the core to actually receive the interrupt (order between the unblock and interrupt generation doesn't matter)
    - Note that even after triggering Hostboot, SBE must continue deadman timer to check that Hostboot recovers from the master STOP15 cycle. If Hostboot does not stop deadman timer in X seconds (passed in as parameter), SBE must checkstop system. The X seconds is the full time
    - On failure the SBE FW will trigger a checkstop
    - On pending if the timer has expired then trigger a checkstop.

♦ p9_trigger_stop15 – Hostboot path (Hostboot running)

- Hostboot function, not a HW Procedure
- ***p9_block_wakeup_intr.C -set***
    - This will prevent all interrupts/wake up sources to the core, thus allowing the next step (STOP 15) to work
- Hostboot sets up interrupt presenter so OCC ISC port in PSIHB to interrupt master core thread 0
    - If we are in fused – there always be even/odd pair – SBE should have chosen the EVEN EC as the master – responsibility for HB to enforce config

- Thus HB will always interrupt the same thread 0 PIR in fused/normal mode

- Hostboot sets up the stop exit LPCR, HRMOR, MSR values in HOMER based on PIR
  - If in fused mode need to set SPR values into 0,2,4,6 if on even EC (or 1,3,5,7 if on odd EC)
- Issue system call to cause all threads to enter STOP 15. Core will then enter STOP 15 state
  - Clear LPCR (cover not entering due to external interrupts)
  - Write PSSCR with Level = 15,
  - Issue ***stop*** instruction typ

♦ *p9_trigger_stop15_exit* – Cronus path only (Hostboot not running)

- Since Hostboot is not running (cores are all in STOP 15 by default) this procedure will force all cores to exit STOP 15
  - Greg to think about state of the cores after step 4-5
  - This procedure is a NO-OP when the real SBE is executing. It is hook to allow the Cronus to trigger the STOP 15 exit – ie resume execution of the STOP15 flow

♦ Hostboot sends a message to the SBE to exit the deadman loop for exit STOP15

- Hostboot runs when active, otherwise Cronus will have to execute
  - Stops the deadman timer

♦ Hostboot must issue its own IPIs to threads 1-3 (normal) or 1-7 (fused)

16.2 host_activate_slave_cores : Activate slave cores

♦ Hostboot active:

- Setup stack space for all slave core threads –
- Wake up all threads on all cores via IPI commands
  - Cores are sitting in a STOP15 state (flush that way)
  - Issue IPI to all slave threads/cores to force winkle exit. Will start executing at SRESET vector (0x100). Bring them into Hostboot collective
- Enable OCC to collect FIR data on all cores on checkstop
- If the slave cores fail to report call ***p9_dump_stop_info.C*** to collect FFDC

♦ Hostboot not running:

- Cores come alive and into maintenance mode (LPCR not set)

- ***p9_activate_stop15_cores.C* – Cronus path only (Hostboot not running)**
  - Called on a core target
  - SP/Cronus issue IPIs to all cores/threads in system except for those on master core

16.3 host_secure_rng : Secure the random number

a p9_rng_init_phase2.C

- This HWP will check the result of the Random number generator (RNG) diagnostics
- It will also set the RNL security bit to prevent the RNG from being reprogramed via Xscom by the hypervisor

16.4 mss_scrub : Start background scrub

a p9_mss_scrub.C (mcbist) – Nimbus

b p9c_mss_scrub.C(mba) -- Cumulus

- Note that this is not executed directly by Hostboot (instead triggered by PRD), Cronus will execute HWP directly
- Start background scrubbing in a continous 12h scrub cycle
- Currently Hostboot will not wait (block) before flowing out to memory
- The completion of the scrub commands must be handled by Host based PRD
- HostPRD will not be called after this point (not called for this step)

16.5 host_load_io_ppe : Load various IO PPEs on each chip

a p9_io_obus_image_build.C(obus pervasive chiplet target, pointer to HCODE ref image)

- For each functional obus load the Nvlink image into the PPE SRAM (32KB image)
  - Sequence of scoms
  - Can load regardless of Nvlink/OpenCAPI. Will sit “idle” until triggered by NVLink DD
  - No planned usage of image for OpenCAPI
- This may be done in parallel for all o/x bus units for a performance optimization
- After the image is loaded this HWP will start the PPE and check that it is running

b p9_io_xbus_image_build.C(xbus pervasive chiplet target, pointer to HCODE ref image)

- For each functional xbus chiplet load an image into the PPE SRAM (64KB image)
  - Sequence of scoms
  - No planned usage for product, lab usage only
- This may be done in parallel for all o/x bus units for a performance optimization

- After the image is loaded this HWP will start the PPE and check that it is running

16.6 host_ipl_complete : Notify SP drawer ipl complete

- Stop hostPRD (in anticipation that HBRT will take over PRD responsibilities)

b Sends a message to SP that drawer IPL is complete

- Pushes down all attributes
- Hostboot enters a "quiesced" state
- Setup any data structures/locks for potential drawer merge
- Sends asynchronous trigger message to the SP indicating that this step is done on this drawer and SP should proceed with the IPL. This message is **not** sent in istep mode
- At this point the SP takes over the IPL

17 Step 17 SP – Init PSI

-

18 Step 18 Establish System SMP & TOD

18.11 proc_tod_setup

- On FSP Based systems this is run on the FSP, BMC based systems this is run in Hostboot
- FW owns algorithm of TOD topology, HWP pushes values into HW

b p9_tod_setup.C

- FW passes in a topology tree, which TOD oscillator to use, and primary/secondary topology
- HWP determines delay values from attributes (MRW)
- HWP programs HW
- HWP outputs register values needed for PHYP and PRD analysis

18.12 proc_tod_init

- On FSP Based systems this is run on the FSP, BMC based systems this is run in Hostboot
- Performed to init the TOD network. Done during the FW IPL due to AVPs, note that it will be done again by PHYP when they start

b p9_tod_init.C

- Setup EX chiplet TOD

19 Step 19 SP – Prepare for Host

20 Step 20 Hostboot – Load Payload

20.1 host_load_payload : Load payload

- build_host_data : Build the host data areas
  - This step builds the HDAT data areas from attributes, VPD, etc
- Load payload. This can either be directly from PNOR (controlled by attribute) or via the SP
  - PNOR path – just loads what is in payload section on flash
  - SP path
    - When the Host sent the complete IPL message for host_ipl_complete part of the payload is the address to load PHYP at (along with a size)
    - For initial BU (non secure mode) PHYP will be loaded via raw DMAs
    - For secureboot PHYP must be loaded via TCEs
    - Payload will be placed in memory based on Hostboot attributes
      - Base address is defined by ATTR_PAYLOAD_BASE When Payload is started this is the HRMOR
      - Starting address is defined by ATTR_PAYLOAD_ENTRY
      - HDAT is placed at well known address off of the image start address
      - **All addresses must be security checked by Hostboot before starting payload**
      - Hostboot then performs verification on the payload

21 Step 21 Hostboot – Start Payload

21.1 host_runtime_setup

- Note that this step is only issued to master HB instance
- Take down any/all TCE setup
- Loop through attributes and write them to predefined memory area inside of the HDAT structures
  - Note: HB master issues IPC to HB slaves for them to update their sections

- Append the TPM log to HDAT structures
  - Note: HB master issues IPC to HB slaves for them to update their sections
- In AVP mode Hostboot will load the OCC and start it here. If the load/start fails then HB will send a errorlog to the SP and the SP will terminate the IPL
  - OCC must monitor for the broadcast scom read (OR) of EX scratch register 7 for the removal of the payload started signature before using the FSI2Host mailbox for ATTN traffic. Note that OCCs on non master chips will never have to wait (as Hostboot only uses the FSI2Host mailbox on the master chip)

21.2 host_verify_hdat

- Only issued to master HB instance
  - If needed IPC to slaves to perform their tasks
- Secureboot verification of PHYP/AVP image load

21.3 host_start_payload

- Prior to starting shutdown sequence Hostboot must write hostboot (ASCII) to scratch register 7 on the master core. All other cores on the master chip must be written to same value or 0s. This value will be polled by the SP in the next step to ensure that hostboot has truly quiesced
- Hostboot enters shutdown sequence
  - Quiesce mailbox and all DMAs
  - Flush data to PNOR
  - Disable interrupts
  - Send sync message to SP (or respond to istep)
  - Enter Kernel
  - Prepare to jump to payload – at this point hostboot must not TI
  - Clear scratch register 7 on master core
- Payload is started by
  - switching HRMOR to desired address and jumping to entry point
  - Note that master thread must be the last one to jump
  - payload cannot start until all threads have been transitionedFor multi-node systems the HB master does the following:
    - Issue slave node shutdown request via IPC
    - HB master polls the “Hostboot done scratch reg” for all slave nodes to enter payload

- HB Master issues own shutdown

♦ No Hostboot code is reused, only mechanism is data passed in HDAT areas. Hostboot runtime is a separate binary image

4 Host Services

The following are not IPL time procedures, but functions called by PHYP/OPAL on Hostboot to perform various tasks. The numbering has been kept common (for convention), but they are not guaranteed to run in this order

State at this point

- PHYP/OPAL running
- Memory is initialized
- SMP alive
- All cores have gone through winkle and are running

22 Enable STOP15

This step is controlled and issued by PHYP when they are ready to build the STOP image. It is required that the STOP image be present in memory prior to loading and starting the OCC.

22.1 host_build_winkle : Build runtime winkle images

♦ Pull Reference Image from SP or PNOR

- Run through secure boot algorithm

b P9_hcd_image_build.C

♦ Called for each processor chip.

♦ Parameter: Pointer to Reference image.

♦ Parameter: Pointer to Output HOMER location. The procedure places the respective images (eg SGPE, CME) into HOMER at the appropriate offsets.

- This is any Hostboot specified mainstore location (does not have to be attached to the processor being STOPed

♦ When PHYP is loaded, the HOMER will be trampled, PHYP will call ***p9_hcd_image_build*** to recreate them in a PHYP specified location in mainstore (each image will probably be placed in mainstore local to its associated processor for performance).

♦ Customize image with data for each core

- Scan rings – Time, GPTR, Repair
- Tweak to make runtime acceptable – expect to be only scom registers

♦ Write image to output pointer parameter

22.2 proc_set_homer_bar : Tell OCC complex HOMER loc

a p9_set_homer_bar.C

♦ Called for each processor chip.

- Parameter: Physical address within HOMER image where OCC code will be loaded; STOPGPE image is this value + 1MB (not a pointer address, it cannot be dereferenced)
  - **NOTE**: HOMER is a 4MB region that is allocated to start 1MB ***before*** the value passed to this procedure!! This done to allow the OCC boot from the 0 offset of the PBA BAR0 value (which has a granularity of 1MB while the Core Self-Store portion must be aligned to a 2MB boundary. Additionally, the OCC complex has no need to address the first 1MB of HOMER --- only the last 3MB.
- Parameters: PBA BAR number, OCC Complex HOMERimage size, STOPGEimage location (default: mem; others: L3, SRAM)

♦ ***p9_pm_pba_bar_config.C (called as subroutine)***

- Set BAR address

22.3 p9_stop_gpe_init -init : Initialize the STOPGPE

a p9_pm_stopgpe_init.C chip_target, ENUM:PM_INIT) -> FAPI2::ReturnCode

♦ Called for each processor chip

♦ Parameters: PM_INIT (to perform initialization vs PM_RESET that is used during the OCC reset flow)

♦ Bootloader runs from HOMER OCC offset + 1MB (2MB from HOMER base)

- Copies STOP image from HOMER to OCC SRAM
- Restarts from OCC SRAM
  - PK initialization -> STOP Thread(s) started

♦ Sets flag in OCC Flag reg that initialization is complete for HWP to poll on

♦ Loop over all functional cache chiplets

- ***p9_pm_pfet_init.C (cache target, PM_INIT)*** (called as a subroutine)
  - Initialize PFET controller parameters (delays)

♦ Loop over all functional core chiplets

- ***p9_pm_pfet_init.C (core target, PM_INIT)*** (called as a subroutine)

- Initialize PFET controller parameters (delays)

♦ **NOTE:** CME initialization is performed upon STOP exit of the cache chiplet by the STOPGPE as to allow the wake up of any core within a Quad. This is NOT done via HWPs.

♦ ***p9_stop_gen_cpu_reg()*** will be called by PHYP prior to ***stop***ping any core

- API that updates a STOP image with various chip state registers (MSR, HRMOR, LPCR)
- The chip registers are set to these values on STOP exit
- **This will only be called directly by PHYP at their discretion. Hence separate from *p9_hcd_image_build*.**

23 Reset and Initialize OCC

This step will run each of the substeps to each chip within a physical node (an OCC boundary) before proceeding to the next step. This is done as a regular process in looking to the "start_occ" step whereby the OCCs will start in reasonable time proximity (one followed by the next via singular XSCOM to each chip) to minimize OCC startup timeouts.

23.1 Setup OCC bars : Establish legal addressing

a p9_pm_pba_bar_config.C chiptarget, address

♦ Address dictated by PHYP

♦ Called once for each of 4 BARs

♦ Place image in EM Nodal Region at offset 0

23.2 power_management_reset : Reset Power Management

(includes clearing any latent errors that may be pending; done for the case of OCC reset)

a p9_pm_init.C – reset, chiptarget

- ***p9_pm_firinit & i_chip_target, ENUM:PM_RESET*** : Save the current FIR mask setting for later restoration and then set all masks to keep errors from occurring during the reset and initialization
  - ***p9_pm_ppm_firinit.C &i_chip_target, ENUM: RESET***
    - For all configured EC chiplets, save and set all FIR Masks
    - For all configured EQ chiplets, save and set all FIR Masks
  - ***p9_pm_occ_firinit.C &i_chip_target, ENUM: RESET***
    - save and set all FIR Masks
  - ***p9_pm_pba_firinit.C &i_chip_target ENUM: RESET***
    - save and set all FIR Masks
- ***p9_pm_occ_control.C chiptarget, ENUM:OCC_HALT***

- OCC PPC405 is halted to allow for a clean stop
- Will cause HW heartbeats to cease and HW will enter safe mode (quiese pStateGPE) – expect to take less than 10 ms

- ***p9_pm_occ_control.C *chiptarget, ENUM:OCC_STOP***
  - OCC PPC405 put into reset
- For all configured cores, ***p9_cpu_special_wakeup.C *ectarget, ENUM:ENABLE –entity ENUM:OCC***
  - Not used by PHYP – custom procedure used
  - Uses the OCC special wake-up bit.
    - Doesn't collide with FSP/PHYP bits.
    - Takes the SGPE, CME, OCI and PBA out of the equation
    - Take PPM PFET controller out of the equation
  - Poll for completion.
  - If timeout, indicates that restart of OCC is to not occur via fapi::ReturnCode
    - RC_PROCPM_SPC_WAKEUP_TIMEOUT
    - PRD effect: Mark chiplet for garding
  - Note: SGPE detected errors (which includes CMEs as well) will produce malfunctions alerts to PHYP whereby the set of events defined in ***p9_stop_recovery.C*** occur to deal with getting the idle handling complex recovered for use.
- ***p9_pm_stop_gpe_init.C *chiptarget, ENUM:PM_RESET***
  - Halt 24x7 processing
  - Halt STOP GPE engine
  - With Special Wake-up in place, this engine is not being used.
- ***p9_pm_pstate_gpe_init.C *chiptarget, ENUM:PM_SAFE_MODE***
  - Command the Pstate GPE engine to put the chip into Safe mode
    - If PGPE is operational,
      - Clears “SAFE_MODE_COMPLTE” and sets “SAFE_MODE_IN_PROGRESS” in OCC Flag Register (this gives this procedure positive feedback that PGPE is acting no this request)
      - Use existing Pstate protocols (eg CME Quad Manager assumed operational) to move to nominal frequency and voltage (PGPE has the VPD points to it know where that is).
      - PGPE
    - Else if PGPE is not operational (SAFE_MODE_IN_PROGRESS not set in 10ms timeout or SAFE_MODE is not set in 500ms timeout --- the latter can occur if CMEs are not responsive)

- Read present external voltage using O2S Bridge B
- If voltage is above the safe voltage (eg the voltage need for the safe frequency), read all EQ FREQ_CNTL_REG to determine the present DPLL frequencies.
  - If all are at or below the SAFE frequency, leave;

- ***p9_pm_pstate_gpe_init.C *chiptarget, ENUM:PM_RESET***
- Halt Pstate GPE engine
  - With PPC405 stopped and safe mode in place, this engine is not being used.
  - Halt the engine via XCR[CMD] = "halt"
  - Note: this will engage the PPM "OCC Heartbeat" which will cause the CMEs to move to it's safe frequency
- ***p9_pm_occ_gpe_init.C *chiptarget, ENUM:PM_RESET***
- Halt both OCC GPE engines
  - Set OCC Flags to request a graceful halt, after timeout will force
  - If forced off, then need to relinquish I2C engines (if owned by OCC) and send interrupt via OCC_MISC
  - With PPC405 stopped and safe mode in place, this engine is not being used.
  - Halt the engine via XCR[CMD] = "halt"
  - Note that when the OCC GPEs stop, the Nimbus/Cumulus memory will throttle into safe mode due to lack of polling

- ***p9_pm_corequad_init.C chiptarget, ENUM:PM_RESET***
  - For **all** configured EC chiplets
    - Place holder at this time
  - For **all** configured EQ chiplets
    - Force OCC SPR Mode in each CME to remove OPAL communication path
    - Adjust clock grid for the safe frequency to allow for HOMER updates of clock grid parameters
    - Check that DPLLs are at the safe frequency. If not, move them there
    - Disable (into bypass) the Quad IVRMs to allow for HOMER updates of iVRM parameters
    - OCC Heartbeat disable
      - Will be enabled by pstateGPE Hcode (not FAPI)
  - ***p9_pm_pba_init.C *chiptarget, ENUM:PM_RESET***
    - Issue resets to all 4 PBA Slaves; poll for completion
    - This does not touch the PBA BARs
  - ***p9_pm_occ_sram_init.C *chiptarget, ENUM:PM_RESET***
    - Placeholder
  - ***p9_pm_ocb_init .C *chiptarget, ENUM:PM_RESET***
    - Disable all OCB indirect channels and return them to their power-on state
      - Note, may need to leave one of the channels enabled for SBE<->Host comm
  - ***p9_pm_pss_init .C *chiptarget ENUM:PM_RESET***
  - See that any outstanding operations have finished in ADC engine

        - See that any outstanding operations have finished in P2S engine
    - ***p9_pm_init.C *chiptarget, ENUM:PM_INIT***
        - ***p9_pm_corequad_init.C *chiptarget, ENUM:PM_INIT***
            - Placeholder
        - ***p9_pm_ocb_init.C *chiptarget, ENUM:PM_INIT***
            - Put registers back to their initial settings
        - ***p9_pm_pss_init.C *chiptarget, ENUM:PM_INIT***
            - Setup PSS Configuration (PSS Frequency (attribute) to PSS macro settings)
        - ***p9_pm_pba_init.C *chiptarget, ENUM:PM_INIT***
            - "PowerBus Slave" buffer set configuration. Assigns slaves to OCI masters for runtime (vs IPL time for HBI loading)
            - PBA Configuration
                - Hang pulse dividers
                - Drop priority (MRWB attribute - TBD)
                - Overcommit counter settings (MRWB attribute - TBD)
        - ***p9_pm_firinit.C :*** Set the FIR masks and action bits per RAS FIR spreadsheet; done as FAPIs vs scom.initfiles to be supportable under PHYP
            - ***p9_pm_ppm_firinit.C *chiptarget, ENUM:PM_INIT***
                - Put registers back to their initial settings For all configured EC chiplets, sets FIR Masks and actions registers (first time takes on initial mask value; subsequent calls restores the value saved during ***ENUM:PM_RESET*** into an attribute
                - For all configured EQ chiplets, sets FIR Masks and actions registers (first time takes on initial mask value; subsequent calls restores the value saved during ***ENUM:PM_RESET*** into an attribute
            - ***p9_pm_occ_firinit.C *chiptarget, ENUM:PM_INIT***
                - sets FIR Masks and actions registers (first time takes on initial mask value; subsequent calls restores the value saved during ***ENUM:PM_RESET*** into an attribute
            - ***p9_pm_pba_firinit.C *chiptarget, ENUM:PM_INIT***
                - sets FIR Masks and actions registers (first time takes on initial mask value; subsequent calls restores the value saved during ***ENUM:PM_RESET*** into an attribute

24 Load OCC

24.1 load_occ : Place OCC image into memory

- For each chip in a physical node
- There are two divergent paths to load the OCC code image. The first is lab/Cronus only without FW. In this case the HWP is run. In the second case FW controls building up the image at the direction of PHYP

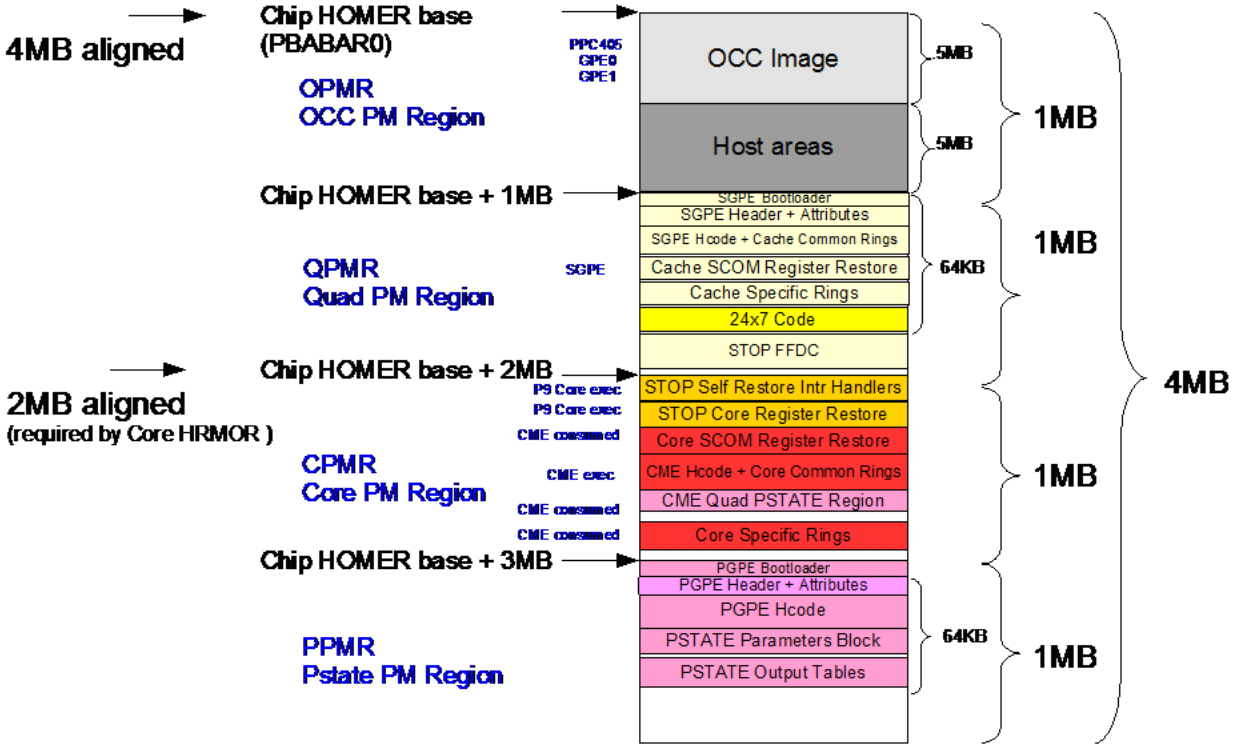
b p9_occ_load.C CRONUS ONLY, mimics what FW does

- Load image in memory from PNOR at an address that is passed to this procedure

c occ_load: FW

- There are four different scenarios where this will get run:
  - PHYP: calls HBRT Adjunct
  - OPAL with FSP: HBRT directly within OPAL
  - OPAL openPOWER: Hostboot calls this prior to starting OPAL
  - AVP mode: Hostboot call this prior to loading AVP
- HBRT called with memory region to place the HOMER image
  - HBRT obtains OCC, reference image
    - FSP based systems via lidmgr
    - OpenPOWER systems via PNOR
  - Entity that loads the image verify signature through secure algorithm
    - Lidmanager PHYP
    - PNOR HBRT
- HBRT will create the STOP image from the reference image (see step 15 of IPL)
  - HBRT will recreate the whole image each time (both OCC/PState,
  - ***p9_hcode_image_build.C*** (void* reference_image, void* v_homer_region, ALL)
    - This includes the SGPE, PGPE, CME.
    - Step 15 built the SGPE and CME components (STOP function)
    - The PGPE is tied to the OCC function
      - Manufacturing request to allow biasing
      - Build Pstate Parameter Block (PPB)
      - Good cores come via the deconfig register
- HBRT will place OCC initial startup information into HOMER image
  - Nest Frequency
  - Interrupt type – FSI2Host mailbox(TMGT) or via PSIHB(HTMGT)
  - FIR Master
  - FIR Capture Data (generated by HBRT) – non FSP based systems
  - Processor map, and FIR register to read

- HBRT places STOP and OCC images as directed by caller. Here is an overview of a completed HOMER layout:



25 Start OCC

25.1 start_occ : Start OCC

*a p9_pm_stop_gpe_init *chiptarget, ENUM:INIT*

- Sets the IAR to the SGPE bootloader in HOMER.
  - HOMER base (PBABAR0 + 1MB) + 16B
- Starts the SGPE and polls OCC Flag bit for HCode init completion
  - Starting the SGPE will cause a “reboot” of active CMEs
    - SGPE will cause Block Copy Engine to pull CPMR code, common quad rings and Core Pstate Parameter Block into CME SRM
    - This will start both STOP and HiPFV(Safety/WOF) and QuadManager (Pstate) threads
    - QM thread will send a PCB Interrupt to PGPE to indicate “ready”
  - SGPE checks that CME STOP functions have started as part of the HCode init complete

- HiPFV(Safety/WOF) and QuadManager (Pstate) check will be done by PGPE upon Pstate protocol start

b p9_pm_pstate_gpe_init *chiptarget, ENUM:INIT

- ♦ Sets the IAR to the PGPE bootloader in HOMER.
  - HOMER base (PBABAR0 + 3MB) + 16B
- ♦ Starts the PGPE and polls OCC Flag bit for HCode init completion
  - Will scoreboard the receive QM ready messages to known which CMEs have QMs
  - Will NOT start Pstate Protocol until commanded by OCC FW

c p9_pm_occ_control.C *chiptarget, ENUM:OCC_START

- Starts OCC load by releasing the reset to the PPC405
- OCC code boot loads itself from Memory into SRAM tank

26 Config OCC

26.1 config_occ : Load OCC config

(H)TMGT now builds the OCC config data and uses its communication path to OCC to give pass config information

a OCC FW sends OCC IPI to PGPE to start Pstate Protocol

- PGPE reads Pstate Parameter Block (PBB) from HOMER, installs in OCC SRAM, and starts the Pstate Protocol with the CMEs.